Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Ambiente de análise de ameaças para geração de Inteligência de Ameaças usando Fontes Abertas

José Valdy Campelo Júnior

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. João José Costa Gondim

Brasília
2018

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Ambiente de análise de ameaças para geração de Inteligência de Ameaças usando Fontes Abertas

José Valdy Campelo Júnior

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. João José Costa Gondim (Orientador)
CIC/UnB

Prof. Dr. Paulo Angelo Alves Resende
CIC/UnB

Prof. Dr. Robson de Oliveira Albuquerque
ENE/UnB

Prof. Dr. Ricardo Pezzuol Jacobi
Coordenador do Curso de Engenharia da Computação

Brasília, 06 de fevereiro de 2018

Dedicatória

Dedico este trabalho aos meus pais, José Valdy e Lúcia, que com muito amor moldaram o homem que sou. À minha esposa, Ericka Pereira, que com sua paciência e apoio tem me ajudado até aqui.

Agradecimentos

Agradeço a Deus por em todos os momentos prover o que eu necessitava. Agradeço a meus pais e minha irmã por todo apoio durante o decorrer do curso.

À minha esposa por todo seu empenho e paciência.

Em especial, agradeço ao professor Caetano por sua presença e paciência sempre sendo um ombro amigo, a meu chefe Luís Cláudio por todo suporte e amizade e a Felipe Borges por sua amizade e auxílio para a realização deste trabalho.

À minha madrinha de curso, a servidora Maria Helena cuja, ajuda foi imprescindível para conclusão deste trabalho.

À minhã amiga/irmã Raquel Ingrid que esteve ao meu lado desde o começo e nas maiores dificuldades.

Aos meus amigos Ana Carolina, Lucília Oliveira, Jorge Mendes, Stéphannie Chiang, Rafael Bispo, Matheus Ervilha, Eduardo Schuabb e Heitor Queiroz.

Aos meus amigos de trabalho Aline dos Santos, Camila Camargo, Christian Luis, Caio Oliveira, Mariana Makiuchi, Arthur Veiga e Jônatas Gomes.

Ao meu professor orientador João Gondim por todo o conhecimento compartilhado e apoio.

Resumo

A análise de ataques a redes de computadores é complexa dado o volume de dados trafegados e a quantidade elevada de máquinas, mesmo em pequenas redes. O volume de dados é grande e o tempo para processá-los e analisá-los é curto. Todo dia uma nova ameaça ou um novo ataque são criados e dissipados na Internet.

O trabalho teve como o objetivo utilizar ferramentas de particionamento não supervisionado para análise e extração de informações, que foram obtidas de fontes abertas. Inicialmente os dados foram obtidos do site da *Norse Corporation*. Resultados significativos foram obtidos, informações sobre ataques em portas com valores acima de 50.000 que são realizados principalmente por atacantes chineses. Corroborando essa informação várias empresas reportaram a existência de *backdoors* nessas portas em roteadores de fabricantes chinesas.

Devido a limitações processo, que era centralizado e utilizava técnicas tradicionais, o objetivo evoluiu. O foco agora era desenvolver uma arquitetura robusta, elástica e escalável que faz uso de técnicas de processamento com o uso do Hadoop para que assim as informações sejam disponibilizadas em tempo hábil. Dessa forma novas fontes de dados sobre ataques puderam ser incluídas de forma a diversificar as informações presentes na base de dados.

Com mais de 20 milhões de ataques no sistema uma análise muito mais profunda pode ser realizada, apresentando uma série de resultados animadores. Além disso, o objetivo foi atingido, a arquitetura proposta foi implementada e possui todas as características desejadas processando os dados em tempo quase real.

O sistema provê informação de inteligência sobre ataques em grande escala com agilidade e eficiência. Sua utilização é proveitosa em diversas organizações que baseiam seu funcionamento em sistemas computacionais, e ainda para nações que desejam proteger sua infraestrutura de forma pró-ativa.

Palavras-chave: detecção de ataques, clusterização k-means, inteligência de ameaças, big data

Abstract

The analysis of attacks on computer networks is complex given the data volume transmitted and the high number of machines even in small networks. The data volume is large and the time to process and analyze them is short, since every day a new threat or a new attack is created and dissipated in Internet.

The purpose of the work was to use unsupervised partitioning tools for analysis and extraction of information, which were obtained from open sources, initially from the *Norse Corporation* website. Significant results were obtained, information of attacks on doors with values above 50.000 that are carried out mainly by Chinese attackers. Corroborating this information several companies reported the existence of *backdoors* in these ports on routers from Chinese manufacturers.

Due to process limitations, which was centralized and used traditional techniques, the objective has evolved. The focus was now on developing a robust, resilient, and scalable architecture that makes use of Hadoop processing techniques so that information is made available in a timely manner. New sources of attack could be included in order to diversify information present in the database.

With more than 20 million attacks on the system a much deeper analysis can be performed, presenting a number of encouraging results. In addition, the objective has been achieved, the proposed architecture has been implemented and has all the desired characteristics, processing the data in near real time.

The system provides intelligence on large-scale attacks with agility and efficiency. Its use is beneficial in many organizations that base their functioning on computer systems, and also for nations that wish to protect their infrastructure in a proactive way.

Keywords: attack detection, k-means clustering, threat intelligence, big data

Sumário

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

API *Application Programming Interface.*

AS Sistema Autonomo.

ASN *Autonomous system Network.*

CN República Popular da China.

DARPA *Department of Defense's Advanced Research Projects Agency.*

DDoS *Distributed Denial of Service.*

DoD *Department of Defense.*

GSS-API *Generic Security Standard Application Programming Interface.*

HDFS *Hadoop Distributed File System.*

HTML *HyperText Markup Language.*

HTTP *Hypertext Transfer Protocol.*

HUE *Hadoop User Experience.*

IANA *Internet Assigned Numbers Authority.*

IESG *Internet Engineering Steering Group.*

ISP *Internet Service Provider.*

JSON *JavaScript Object Notation.*

MOTD *Message of the day.*

MULTICS *Multiplexed Information and Computing Service.*

OTAN (NATO) *North Atlantic Treaty Organization.*

PDTI Plano Diretor de Tecnologia da Informação.

RAM *Random Access Memory.*

RDD *Resilient Distributed Dataset.*

RFB *Remote Frame Buffer.*

SMB *Server Message Block.*

URI *Uniform Resource Identifier.*

US Estados Unidos da América.

VNC *Virtual Network Computing.*

YARN *Yet Another Resource Negotiator.*

Capítulo 1

Introdução

O domínio da informação é ponto fundamental para a evolução da humanidade desde seu início. O desenvolvimento da escrita e da matemática permitiu às sociedades perpetuarem seu conhecimento aumentando o volume de conhecimento total da população. Esse aumento permitiu a esses povos desenvolver ferramentas que permitiram modificar o ambiente a sua volta, de forma a melhorar sua qualidade de vida e as interações humanas.

A partir da revolução industrial, a informação ganhou uma importância ainda maior, pois conhecer o processo e os custos de produção permitia as empresas aumentar sua eficiência e se colocar em posição privilegiada perante a seus concorrentes. Com um mundo iniciando sua globalização duas grandes guerras romperam, aumentando ainda mais a relevância da análise das informações.

As nações durante a guerra desejam obter informações sobre seus inimigos de forma a prever possíveis ataques, para que assim medidas protetivas ou reativas pudessem ser tomadas. Por outro lado, necessitavam de mecanismos robustos e seguros para transmitir suas próprias informações de forma que o inimigo não conseguisse acessá-las.

Durante a guerra, principalmente em seu fim, o desenvolvimento de uma máquina capaz de processar e analisar informações em uma velocidade nunca antes pensada foi acelerada. O computador permitiu que um volume de informações muito grande de dados pudesse ser tratado, processado e analisado.

Porém, como no mundo real, ameaças afligem o universo computacional de forma igual ou até mesmo com mais intensidade. Atualmente, grande parte das empresas têm suas operações totalmente dependentes de sistemas computacionais, sejam para executar sua atividade fim ou para auxiliar em sua execução.

Garantir que esses sistemas estejam seguros e que suas operações não sofram ataques com a intenção de paralisá-los, ou ainda de extrair informações confidenciais, é o desafio da Segurança Computacional da Informação. Porém esta não é uma tarefa fácil, com a

Internet o número de ameaças é gigantesco, um atacante pode de sua casa subverter um sistema causando graves prejuízos.

Conseguir analisar dados sobre ameaças aos sistemas é imprescindível para estruturar planos de proteção e recuperação. Entretanto garantir que seja possível obter essas informações completas e em tempo hábil é um desafio que se apresenta na atualidade.

1.1 Objetivos

1.1.1 Objetivo Geral

Este trabalho busca a captura, estudo e análise de ataques em rede obtidos através de fontes abertas. Dessa forma, deseja-se gerar informação de inteligência sobre esses para proporcionar um melhor conhecimento de posicionamento das corporações frente a estas crescentes ameaças.

1.1.2 Objetivos Específicos

Para alcançar o Objetivo Geral apresentado, este trabalho define uma arquitetura robusta para extração e análise de informações a partir de um conjunto massivo de dados. Essa análise e extração devem ser executadas de forma que os resultados sejam obtidos em velocidade muito próxima da obtenção desses dados.

As técnicas e ferramentas utilizadas para compor a estrutura dessa arquitetura dispondo de ferramentas difundidas no mercado mundial, de forma que essa se mantenha sempre atual e comportando o volume de dados de forma escalável. Dessa forma, grandes modificações não serão necessárias, mantendo o custo humano focado em extrair informações para a construção de um conhecimento que permita, em última instância, prever a ocorrência de ataques com tempo suficiente para evitá-los.

1.2 Organização do trabalho

O trabalho está estruturado da seguinte forma: o Capítulo 2 apresenta os conceitos básicos e as ferramentas utilizadas durante a realização deste. A abordagem desses conceitos busca permitir ao leitor completo entendimentos da proposta. No Capítulo 3 são apresentadas as duas abordagens realizadas para alcançar os objetivos do trabalho com descrição detalhada de seu projeto e implementação. O Capítulo 4 trás os resultados das duas abordagens realizadas, bem como uma breve análise sobre esses. Por fim, o Capítulo 5 apresenta uma breve conclusão sobre os pontos deste trabalho e também ideias de trabalhos futuros para a complementação deste.

Capítulo 2

Conceitos básicos e ferramentas utilizadas

Neste capítulo são apresentados os conceitos básicos e principais ferramentas referidas neste trabalho.

2.1 Conceitos básicos

Os conceitos básicos buscam situar o leitor a cerca do tema explorado por este trabalho, bem como uma perspectiva histórica sobre o assunto.

2.1.1 Informação

Informação é uma abstração informal (isto é, não pode ser formalizada através de uma teoria lógica ou matemática), que representa algo significativo para alguém através de textos, imagens, sons ou animação [1]. Esse conceito, entretanto, não é uma definição rígida mas sim um conceito abstrato. Por exemplo, a pintura de um artista é uma informação desde que seja vista ou sentida por uma pessoa que a compreenda e consiga extrair dali um significado. Ou seja, a informação não é uma grandeza única para todo indivíduo, o que pode acrescer ao conhecimento de uma pessoa pode não ser nem compreendido por outra.

Processar diretamente informação em um computador não é possível. Primeiramente é preciso transformá-la e reduzi-la a dados. Dado é definido como *uma sequência de símbolos quantificados ou quantificáveis* [1], ou seja, textos, imagens, sons, já que esses podem ser quantificados seguindo algum tipo de codificação. Em um computador, toda informação é invariavelmente representada por meio de bits, já que não existe outra forma de representação.

Vale salientar que no armazenamento do computador não constam informações, mas sim sua representação na forma de dados. Essa representação pode ser alterada pelo computador através da construção de programas, nos quais sequencias de bits podem ser interpretadas a fim de possibilitar e exibição desses dados de forma mais legível, como por exemplo letras e imagens em uma tela. Entretanto, o significado daqueles dados não é alterado, visto que o significado é diretamente relacionado à pessoa que esta sendo exposta a informação.

Desta forma, o significado da informação é diverso e dependente de cada indivíduo, que através da associação de seus conhecimentos e experiências a um conceito real, tal como velocidade ou posição geográfica, extrai dali um significado.

Mesmo não sendo possível definir formalmente a informação, vários autores se debruçaram sobre elas, provavelmente o mais importante deles foi Claude Shannon (1948) [2] que realizou estudos para quantificar a informação. A principal definição realizada por Shannom foi a de descrever a entropia da informação. Entropia é a medida de incerteza de uma variável aleatória, dada pela equação

$$H(x) = -\sum_{i} P(x_i).log(P(x_i)), \tag{2.1}$$

em que x_i variando em todos os possíveis valores de x e $P(x_i)$ a probabilidade de ocorrência de x_i.

Com o uso da Equação (2.1), é possível, a partir da probabilidade de uma serie de eventos, definir a codificação ótima para esses. Essa capacidade trouxe importantes avanços no campo da compressão, transmissão e armazenamento de dados. Além dessas, sua aplicação foi difundida e atualmente é utilizada em diversos áreas de tecnologia, como por exemplo criptografia, física quântica, biologia, entre outras.

Um dos pontos revolucionários da teoria de Shannon foi sua aplicação em sistemas criptográficos [3], que segundo James Gleick [4], foi a "... primeira vez puderam dispor de uma forma rigorosa de avaliar o grau de segurança de qualquer sistema de sigilo".

Um ponto fundamental é compreender a diferença clara entre dados e informações. Os dados são puramente símbolos utilizados para representar um determinado objeto ou ideia, eles são meramente sintáticos. As informações, por sua vez, têm significado implícito, possuindo semântica. Em um computador, as informações só podem ser obtidas após o processamento dos dados armazenados, porque estes estão em forma não legível para seres humanos. Assim, os computadores não são capazes de entender os dados neles armazenados e extrair significado deles [5].

Como já mencionado, a importância da informação não surgiu com o desenvolvimento dos computadores, sendo esses somente mais uma ferramenta utilizada para armazenar e

processar informação.

O desenvolvimento e evolução da escrita é uma das primeiras estruturações da informação. Antes do desenvolvimento da escrita, grande parte do conhecimento era perdido, isso ocorria por que a única forma de persistir o conhecimento era através da forma verbal e de pessoa para pessoa. Com a escrita, foi possível reproduzir em meio físico as informações, permitindo que mais pessoas tivessem acesso a elas, pessoas de outras gerações que nunca tiveram contato direto com ninguém que tivesse presenciado aquele fato [6].

Porém, no inicio, o único meio de se perpetuar as informações através da escrita era através dos escribas, porque não era de conhecimento geral a técnica e as informações que permitiam escrever. Essa forma de reprodução era ineficiente e também representava um risco ao conteúdo, visto que quem escreve decide o que seria escrito. O primeiro grande salto dado na difusão da informação foi a invenção da prensa de tipos móveis inventada por Bi Sheng na China entre 1041 e 1048 e mais tarde aperfeiçoada por Gutemberg, que a popularizou por volta de 1439 [7]. Após isso, a informação poderia ser massificada, se tornando esse o alicerce no desenvolvimento tecnológico da humanidade.

O primeiro livro impresso na máquina de Gutemberg foi a Bíblia, logo após a reforma protestante liderada por Martinho Lutero. O objetivo dele era disponibilizá-la de forma impressa para toda a população, entretanto a Bíblia se encontrava escrita em latim sendo necessária sua tradução para o alemão. A ação de Lutero serviu para disseminar de forma vasta informação e conhecimento acerca das escrituras, fortalecendo suas crenças em sua doutrina, e ainda teve papel importante para evolução da língua alemã como um todo.

Se por um lado essa ação trouxe grandes ganhos a população daquele período, por um outro ela também mostrou o quanto a negligência com a difusão da informação poderia ser prejudicial. Como por exemplo com uma edição da Bíblia impressa em 1631 por conta de um erro tipográfico foi publicada com um erro, a palavra "não" do nono mandamento foi esquecida, sendo este impresso como "Cobiçarás a mulher do próximo". Esta versão ficou conhecida como A Bíblia Maldita (**The "Wicked" Bibles**) [8]. O erro custou aos responsáveis uma multa alta para o período além da anulação do direito de imprimir a Bíblia, porém o maior problema gerado foi o comprometimento da imagem dos editores com o povo.

Esse acontecimento exemplifica como um erro simples durante a gerência da informação pode ter efeitos graves e duradouros. Dessa forma sendo necessária uma abordagem madura sobre a acurácia da informação gerada e divulgada.

Toda essa evolução das tecnologias da informação permitiu que mais conhecimento fosse acumulado ao longo dos anos. Com isso vários fenômenos aconteceram, como por exemplo, a revolução industrial. Nesse período, houve uma mudança radical na forma de produção. Esse aumento trouxe a necessidade de estudar o processo de produção para

aumentar a lucratividade dos negócios. Desde então, o ser humano vem estruturando as informações de forma que o acesso a ela seja rápido, simples e com alta confiabilidade.

A revolução industrial não foi um acontecimento com data única, mas sim um processo que durou algo em torno de 80 anos, ocorrendo entre os anos de 1760 e 1840 [9]. Sendo um marco do fim do período feudal, a revolução industrial iniciou uma nova forma de produção. Com o desenvolvimento da máquina a vapor e os avanços na siderurgia, a produção em larga escala foi favorecida. Dessa forma, o acesso aos produtos em todo o mundo foi ampliado, transformando o modo de vida da população.

Durante esse período, outra importante evolução na gestão da informação, talvez a mais importante, foi o surgimento da contabilidade de custos e da contabilidade gerencial [10]. A produção baseada em máquinas evoluía de forma substancial, logo o controle de custo se tornava complexo, dado principalmente pelo alto volume e complexidade da produção. Possuir controle eficaz sobre esses custos, bem como dos impactos nos resultados das empresas, se tornou um diferencial para as empresas e seus sistemas de produção, mantendo sua rentabilidade e permitindo que as empresas permanecessem viáveis comercialmente. Edson Pamplona [11] cita que as primeiras organizações a desenvolverem sistemas de contabilidade gerencial foram as tecelagens de algodão nos Estados Unidos, aproximadamente em 1812.

As chamadas máquinas-ferramentas são máquinas utilizadas na produção de outros materiais através da movimentação mecânica de seu conjunto de ferramentas, como por exemplo, um torno mecânico. O desenvolvimento dessas máquinas aumentou a produção e por conseguinte a comercialização de produtos. Por conta disto o volume de informações envolvido no processo aumentou gerando problemas de gestão de informação.

No ano de 1858, o americano Charles Field e os britânicos Charles Bright com os irmãos John e Jacob Bret fundaram uma empresa pra lançar um cabo submarino entre os Estados Unidos e a Inglaterra [12]. Após várias tentativas, o cabo conseguiu ser instalado. Entretanto, poucas semanas depois de ser inaugurado o cabo falhou. Mesmo após o fracasso, oito anos depois uma solução definitiva foi instalada permitindo a comunicação intercontinental. Dessa forma, o tempo que se levava para transmitir uma informação entre os dois países, que antes era de dez dias, caiu para uma questão de minutos. Agilizando a propagação das informações para os processos comerciais e de produção.

Com essa grande quantidade de informações, novas técnicas para gerí-las tiveram que ser desenvolvidas e as antigas aprimoradas, tornando a recuperação e interpretação da informação um processo mais produtivo. Os processos de inventário estão entre os primeiros que utilizaram meios para gerir a informação. Esses processos adotaram várias técnicas como a utilização do *Kardex* [13] e do *Kanban* [14], que permitiam obter as informações de custos do estoque através de fichas de papel.

A partir de 1970, devido ao avanço da eletrônica, o armazenamento e processamento eletrônico de informação popularizou-se aumentando de forma massiva as informações disponíveis. Isso permitiu melhor qualidade das decisões das organizações.

Com a globalização tendo seu início por volta de 1978, as transações comerciais entre nações aumentaram significativamente causando impacto na produção dessas companhias. Mesmo com um sistema estruturado para a gestão da informação, esse não se encontrava livre de falhas. A principal delas era sua limitação na quantidade de informações que poderiam ser armazenadas e manipuladas. Se esse volume fosse muito alto, a velocidade para se manusear e recuperar esta informação diminuía, bem como era necessário um aumento na mão de obra para manipular essas informações. A popularização do computador na década de 80 transformou essa situação. O computador permitia manipular uma quantidade muito maior de informações com um menor número de funcionários e ainda com maior precisão, velocidade e segurança, diminuindo assim os custos das empresas durante o processo de tomada de decisão.

Por consequência desta evolução, o processo produtivo e a eficiência das empresas teve um aumento significativo. Isso levou a uma queda nos custos de produção o que desencadeou uma diminuição nos preços de mercado, permitindo à grande parte das pessoas uma maior gama de produtos e facilitando seu acesso a esses. Porém, na maioria dos casos, os sistemas de comunicação entre os computadores ainda não tinha se desenvolvido completamente ou era inexistente ou rudimentar. Isto mudou quando a popularização da Internet permitiu a conexão barata e com agilidade, integrando várias pessoas ao redor do mundo, o que também mudou o cenário econômico mundial.

Toda essa evolução não trouxe somente progresso. Pessoas mal intencionadas poderiam fazer uso das tecnologias com o objetivo de prejudicar empresas e organizações.

2.1.2 Histórico da Segurança Computacional da Informação

A Segurança da Informação já era desempenhada em outros períodos. O primeiro relato de seu uso data de 1900 a.C., quando um escriba egípcio usou hieróglifos fora de ordem [15]. A partir daí, em várias outras ocasiões a criptografia foi utilizada como meio de se transmitir uma mensagem secreta em um meio não confiável.

Dado o objetivo deste trabalho, o foco será fornecer um breve histórico da Segurança Computacional da Informação. A necessidade desse campo é estreitamente ligada com a origem do computador durante a Segunda Guerra Mundial, quando os primeiros equipamentos foram desenvolvidos para uso em comunicações e quebra de códigos de criptografia.

Vários níveis de segurança foram implementados durante esse período. Contudo, eles eram compostos predominantemente de segurança física sobre os dispositivos e a informa-

ção. Um dos primeiros problemas com a segurança dos sistemas computacionais que foi documentado, ocorreu em meados de 1960, quando um administrador de sistemas estava escrevendo a *Message of the day* (MOTD), que é uma mensagem que seria lida de um arquivo e exibida na tela do usuário sempre que houvesse um login. Ao mesmo tempo, outro administrador estava editando o arquivo de senhas. Por uma falha no software de edição temporária, misturou os dois arquivos fazendo com que todas as senhas fossem exibidas aos usuários que realizassem o login [16].

Durante o decorrer da Guerra Fria, o número de mainframes[1] aumentou para que fosse possível a realização de tarefas mais complexas. Esses computadores necessitavam de um processo de comunicação menos complicado, do que gravar fitas magnéticas e as transferí-las entre os computadores.

Foi para solucionar esse problema que a *Department of Defense's Advanced Research Projects Agency* (DARPA) começou a desenvolver uma rede redundante de comunicação [18]. Nessa época de tensão, existia uma grande demanda dos órgãos militares por transmitir grandes volumes de informação. Outro requisito era o de redundância da rede, já que se existia um grande receio de que uma guerra se iniciasse a qualquer momento. Assim, em 1968, o Dr. Larry Roberts desenvolveu o projeto *ARPANET*, fundando assim o que hoje conhecemos como Internet [19]. A *ARPANET* se popularizou durante a década seguinte, aumentando assim o potencial de problemas que poderiam surgir. Em 1973, Robert M. Metcalfe identificou um grande problema com a *ARPANET*, sua falta de segurança. Ele identificou e elencou uma série de problemas que afetavam o projeto que são vulnerabilidade de senhas, falta de procedimentos de segurança para estabelecimento de conexão, falta de mecanismos de autenticação e autorização de usuários, dentre outros [19].

Nos anos seguintes, vários estudos foram conduzidos para identificar e propor soluções para os problemas da *ARPANET*. O primeiro artigo que propunha outros modos de se proteger um computador além da proteção física foi o *Rand Report R-609* [20], patrocinado pelo *Department of Defense* (DoD) americano. Ele tentava definir múltiplos mecanismos de controle necessários para proteção do processamento de dados em um computador.

Esse artigo agora incluía o reconhecimento de que era necessário proteger os dados em processamento, limitar o acesso aos computadores e aos dados e envolver todos os funcionários de uma organização em uma cultura de Segurança da Informação. Naquela época, o único sistema operacional que levava em conta essas considerações era o *Multiplexed Information and Computing Service* (MULTICS). Ele mais tarde foi descontinuado e substituído pelo UNIX, esse não possuía toda a gama de proteções de segurança de seu

[1]Um mainframe é um computador de grande porte dedicado normalmente ao processamento de um volume enorme de informações. O termo mainframe era utilizado para se referir ao gabinete principal que alojava a unidade central de processamento nos primeiros computadores [17].

antecessor, porém era muito mais simples e rápido de se utilizar, sendo mantido desde aquele período como um projeto de código aberto.

A partir dos aos 90, a popularização dos computadores pessoais e a difusão do acesso à Internet permitiram que o número de computadores saltasse assustadoramente. Com isso, a quantidade de dados gerados também aumentou proporcionalmente, mesmo com os mecanismos de segurança evoluindo de forma constante, as contravenções em sistemas de computadores continuam presentes.

A segurança computacional atualmente é uma preocupação constante de todas as grandes organizações e nações do mundo. O que era antes um tema secundário e muitas vezes deixado de lado, no presente ocupa grande parte das áreas de pesquisa.

Para o desenvolvimento de qualquer sistema computacional é importante primeiro definir os seus requisitos. Com o desenvolvimento de Sistemas de Segurança da Informação não é diferente. É preciso primeiro definir quais são as características críticas da informação, sendo que esta definição não é consenso entre os estudiosos da área. O único consenso é sobre as características da tríade CIA [21], são elas:

Confidencialidade - Segundo a ISO/IEC 27.002 [22], a definição de confidencialidade pode ser expressa por "garantir que a informação seja acessível apenas àqueles autorizados a ter acesso". Esse é o principal aspecto da Segurança da Informação, sendo por muitas vezes o primeiro e mais lembrando.

Integridade - Dizer que uma informação é íntegra significa garantir que ela é corretamente apresentada a quem a consulta. Para isso, deve-se garantir que durante o armazenamento e transmissão da mensagem não há qualquer modificação em seu conteúdo.

Disponibilidade(*Availability*) - É o aspecto que define que a informação sempre deva estar disponível quando for necessária para quem precisar e tiver autorização para visualizá-la.

Outras características que podem ser atribuídas a informação, segundo o livro *Principles of Information Security* [19], são:

Precisão - Quando a informação se encontra livre de erros e tem valor para o usuário final que a espera.

Autenticidade - Quando a informação é genuína, original. Sendo assim, deve possui sempre o mesmo estado desde sua criação.

Utilidade - A informação deve ser útil, ou seja, ter valor para alguém em algum momento. Ela deve sempre servir a um propósito.

Posse - É a característica da informação que lhe atribui propriedade, dando assim a alguém os direitos sobre aquela informação.

2.1.3 Ataques

No universo computacional existe uma vasta gama de ameaças e ataques que podem ser utilizados para infringir dano a um ativo organizacional, impedir o acesso a informação ou subtraí-la. Enquanto na literatura já existem documentadas várias delas, a cada dia novas ameaças são desenvolvidas e utilizadas. Entretanto, para fins de entendimento, todas as ameaças citadas nesta monografia terão suas características abordadas nesta subseção.

Começando primeiro a definir o que é ameaça. Segundo Whitman M. e Mattord H [19], ameaça é uma categoria de objetos, pessoas ou outras entidades que representam um risco ao ativo. Ameaças podem se apresentar como intencionais ou não intencionais.

Dessa forma, a primeira ameaça aos sistemas computacionais era a substração do equipamento ou de seus componentes. Com a evolução das formas de transmissão e processamento dos dados, as ameaças também evoluíram. Passando por ameaças que podiam ser encontradas em discos magnéticos flexíveis até ameaças que se replicam e autopropagam, assim identificar as ameaças e acompanhar sua evolução se tornou uma disciplina obrigatória em segurança.

As principais ameaças que afligem computadores hoje em todo o mundo são apresentadas a seguir [23].

DoS/*Denial of Service* - Ocorre quando o sistema não consegue prover um serviço a um usuário que esteja o solicitando. Isso pode ocorrer por uma série de fatores, retirada do sistema para manutenção, falta de acesso a rede ou ainda de forma má intencionada quando um atacante sobrecarrega a capacidade do sistema. Com a sobrecarga o sistema permanece respondendo a solicitações que tem por objetivo somente lhe ocupar, deixando de responder a um usuário legítimo.

Quando o atacante só dispõe de uma máquina ou quando o ataca parte somente de um pequeno conjunto de máquinas a ameaça é do tipo DoS. Quando um volume gigantesco de máquinas distribuídas por vários países a ameaça se chama DDoS, estas máquinas geralmente são infectadas e controladas de forma remota sem conhecimento do usuário.

Exploit Tools - Ferramentas sofisticadas que podem ser compradas por atacantes com vários níveis de experiência. São utilizadas para detectar vulnerabilidades e conseguir acesso aos sistemas alvo.

Misauthentication - Ocorrem quando um atacantes consegue ingressar em uma rede ultrapassando os mecanismos de autenticação quando eles estão presentes. O atacante utiliza de alguma técnica que convence o sistema de autenticação que ele é um usuário válido e portanto possui acesso para ingressar naquele ambiente.

Sniffer - Programa que intercepta todos os pacotes de uma determinada rede examinando todos eles. É extremamente prejudicial pois pode visualizar todas as informações, como senhas mandadas em texto claro ou mensagens que não estejam criptografadas.

Spam - Mensagens não solicitadas de email com objetivo de enganar o usuário que a recebe, com ofertas que não existem ou informações falsas.

Phishing - Envio de emails que parecem legítimos porém tem como objetivo extrair informações sensíveis do usuário, tais como senhas de banco, senhas diversas, informações de recuperação, entre outros.

Spoofing - Ameaça que envolve falsificar o cabeçalho de uma mensagem, como por exemplo de email, assim a mensagem parece ter sido originado de uma fonte. Porém o atacante que enviou a mensagem para conseguir assim informações sensíveis.

Spyware - Programa de computador que coleta informações pessoais sobre o usuário sem o seu consentimento. Podendo extrair desde letras digitadas no teclado até informações de histórico em navegadores de internet.

Trojan - Programa de computador que contém código prejudicial ao computador, geralmente escondido em um programa que parece ser útil e legítimo.

Virus - Programa de computador que infecta a máquina do usuário. Quando se encontra em memória este programa se replica permitindo ao *virus* infectar outros computadores.

Worms - Programas de computador independentes que se replicam e infectam outros computadores ao longo da rede. Ao contrário dos virus os *worms* não necessitam que um usuário o execute para iniciar suas ações. São utilizados como meio para propagação de outras ameaças.

Zero-Day Exploit - Ameaça que faz uso de uma vulnerabilidade que não foi divulgada, geralmente no mesmo dia. Assim ainda não há nenhuma correção disponível para a ameaça, deixando os usuários completamente expostos.

Botnets - Conjunto de computadores infectados que pode ser controlado remotamente pelo atacante sem o consentimento dos usuários. Esse conjunto é utilizado para diversos ataques, como enviou de *Phishing* ou *Spam*, disseminação de ataques de DDoS, ou lançamento de virus.

O desenvolvimento da Internet e a difusão das redes de computadores desenvolveram um ambiente perfeito para que várias ameaças pudessem se propagar criando uma grande *Botnet*. Antes, o atacante tinha o trabalho de invadir várias máquinas individualmente. Depois de descobrir uma nova vulnerabilidade, ele também tinha de aguardar muitas vezes que estes usuários se conectassem a rede para propagar as ameaças.

Atualmente o panorama é bem diferente para criação destas *botnets*, a maioria dos computadores se encontra 24 horas conectados a alguma rede. Além dos computadores milhares de dispositivos como celulares, câmeras de segurança, até mesmo geladeiras estão conectados diretamente à Internet. Esse é o ambiente para que um usuário mal intencionado prospere. Agora infectando somente um computador, com um *worm* por exemplo, era possível infectar todos os demais presentes na rede.

Para os gestores de segurança e rede, essas mudanças trouxeram grandes dificuldades de gerenciamento. Gerir uma grande organização com milhares de equipamentos e funcionários, exige que uma infinidade de riscos e brechas sejam observadas e monitoradas. Mesmo existindo o presente ferramentas modernas para filtragem de pacotes, detecção de intrusão e antivírus estas não são a bala de prata para os problemas de segurança. Sendo que, a disseminação de uma cultura organizacional em prol da segurança da informação acrescido de treinamentos periódicos aos funcionários tendem a surtir mais efeito.

Outra facilidade que surgiu foi o aumento do número desses equipamentos, sendo que sua grande maioria não esta preparada para suportar as investidas dos atacantes. A falta de atualização, configurações que deixam os sistemas expostos e engenharia social estão entre os principais problemas enfrentados pela maioria dos usuários. Dessa maneira os atacantes conseguem controlar dezenas de milhares de máquinas ao redor do mundo, utilizando delas para o cometimento de outras práticas criminosas sem nunca mostrar sua real identidade.

Nos últimos dez anos várias ferramentas de ameaça em escala global foram notórias, no ano de 2017 por exemplo pode-se citar *Mirai, Hajime, Zeus, NotPetya e WannaCry* causando grandes danos a infraestrutura global da Internet [24, 25, 26, 27]. E essa é uma perspectiva que tende a piorar, o *Mirai* foi a primeira *botnet* a infectar massivamente dispositivos IoT como câmeras de segurança IP, e roteadores. Com a difusão dessa tecnologia de IoT que prega que todos os dispositivos devem estar conectados diretamente à Internet o volume de aparelhos que poderão ser infectados sofrerá um grande salto.

E não são somente atacantes que fazem uso de vulnerabilidades para atividades ilegais, grande parte das nações do mundo enxergam este campo como um pilar da segurança nacional [28]. Temos como exemplo a *North Atlantic Treaty Organization* (OTAN (NATO)) que em Julho de 2017 cogitou em considerar o *malware NotPetya* como um ato de guerra a seus membros [29]. Países como os Estados Unidos vão além e possuem diversas divisões entre suas forças armadas dedicadas exclusivamente a guerra digital, como a 10ª Frota da Marinha Americana que inteiramente dedicada a defesa cibernética [30].

Outros países como a Inglaterra, França, Rússia, Israel e China também possuem grandes esforços neste campo. E os resultados deste investimento já foram recuperados em parte, em 2010 um vírus de computador nomeado como *Stuxnet* foi descoberto pela *Kaspersky Lab* e tinha como principal objetivo destruir usinas de enriquecimento de urânio iranianas [31].

Os principais suspeitos do desenvolvimento e condução desta grande operação ciber militar, foram os Estados Unidos e Israel [32]. E o seu objetivo foi alcançado, visto que após a utilização desta ferramenta o processo de enriquecimento de urânio iraniana foi paralisado já que suas centrífugas sofreram graves danos. Com isso o governo iraniano se viu obrigado a negociar com o Ocidente levando aquela nação a permitir a entrada de observadores internacionais. Desta forma, a capacidade de enriquecer urânio para a construção de uma ogiva nuclear foi praticamente anulada [33].

A Segurança Computacional continua sua evolução em ritmo acelerado e nos últimos anos encontra um novo desafio. Dada a grande quantidade de aparelhos conectados a quantidade de dados também aumentou, se antes era possível abrir arquivos de registro para detectar possíveis problemas ou identificar ameaças na rede hoje isso é impossível. Em um segundo a quantidade de pacotes e requisições que transitam em uma rede é imensa. Esses dados há algum tempo não podem mais ser analisados utilizando ferramentas tradicionais, felizmente um novo paradigma vem ganhando espaço e possibilitando o processamento de volumes de dados inimagináveis anteriormente, este paradigma é explorado na próxima subseção.

2.1.4 *Big Data*

Segundo Tom White [34], “Nós vivemos na era dos dados”. A quantidade de dados armazenados de forma digital não tem um número exato, a IDC projeta que em 2025 serão criados 163 ZetaBytes de dados a serem armazenados em dispositivos digitais. Isso é dez vezes mais a quantidade de dados que foram criados no ano de 2016, 16.1 ZetaBytes [35].

Para colocar em perspectiva essa quantidade de informações, seriam necessários mais de 83 milhões de discos de 12TB para armazenar o número de dados gerados no ano de 2016, e mais de 16 bilhões de disco com esse mesmo tamanho.

Processar e analisar esta quantidade de informações não era possível com uma arquitetura tradicional. Uma forma de contornar estes problemas era retirar somente uma amostra desses dados o que permitia processá-los pelo menos em parte. Essa abordagem felizmente vem mudando com a utilização do paradigma de *Big Data*.

Como definido em *Understanding Big Data* [36], *Big Data* se aplica as informações que não podem ser processadas ou analisadas utilizando processos ou ferramentas tradicionais. E há um aumento considerável de problemas com esta natureza, enquanto por anos o foco dos sistemas era obter a maior quantidade de dados sobre as transações e seus utilizadores, com o crescimento exorbitante do número de usuários dos sistemas, atualmente a dificuldade se encontra em processar e analisar esta quantidade de dados.

Segundo *Understanding Big Data* [36], três características são inerentes ao *Big Data*: volume, variedade e velocidade. O volume diz respeito ao que já foi abordado sobre a grande quantidade de dados que são e serão gerados no futuro, tornando sistemas centralizados impossibilitados de processar todo conjunto de dados.

A variedade diz respeito a diversidade de fontes das quais os dados são originados, registros de acesso de baixo nível, informações sobre localização, acelerômetros, padrões de escrita, entre outros formam a origem das informações. Geralmente esses dados se encontram de forma bruta, não estruturada ou semi-estruturada, tornando assim a análise tradicional inviável visto que seria necessário alterações a cada novo tipo de dado introduzido ao sistema.

Por último, a velocidade para processar esses dados não pode ficar em segundo plano, num ambiente cada vez mais dinâmico alguns segundos podem significar a perda de valor da marca e consequentemente uma perda financeira. Cada sistema, organização ou problema têm um requisito diferente para a velocidade de processamento, entretanto todos eles possuem algo em comum, a velocidade deve permanecer constante ou diminuir a medida que a quantidade de dados cresce. Sendo este um dos pontos fundamentais ao se utilizar a abordagem de *Big Data*.

O processamento de grandes volumes de dados esbarra em alguns problemas que devem ser considerados, como por exemplo diminuir o tempo de acesso e gravação das informações, consultar todos os dados de uma grande base de dados, e realizar a analise interativa de dados.

O armazenamento e recuperação de dados durante o processamento é uma das partes com maior custo temporal para a execução de um programa, quando este processamento envolve uma quantidade de dados que de nenhuma forma pode ser colocada integralmente

em memória esse custo se torna ainda maior. A solução encontrada foi dividir e distribuir os dados ao longo de vários discos e paralelizar o acesso a esses, para reduzir o desperdício de espaço, o acesso a todos os discos seria compartilhado entre os processos e usuários garantindo que na média a utilização desse espaço adicional seja sempre otimizado.

A suíte Hadoop conta com um sistema de arquivos distribuídos o *Hadoop Distributed File System* (HDFS), também conta com uma ferramenta para processamento de grandes quantidades de dados o MapReduce e o *Yet Another Resource Negotiator* (YARN) que é uma ferramenta para controlar todas as tarefas executadas em um *cluster* Hadoop.

Um dos problemas com a divisão dos dados ocorre quando um dos discos falha: qual abordagem deve ser realizada para não permitir a perda permanente daquela parte dos dados. O segundo problema é quanto tempo é gasto para se reagrupar os dados de alguma forma que seja possível realizar o processamento. A solução destes problemas foi implementada pelo sistema Hadoop no HDFS, sendo esse um dos motivos da utilização.

A consulta de todos os dados é realizado pelo Hadoop através do MapReduce, que é detalhado na subseção 2.5.2. Ele abstraí as dificuldades com a agregação dos dados, permitindo ao usuário visualizar os dados como somente uma série de entrada chave/valor. Desta forma, a implementação das soluções se torna muito mais simples, permitindo um maior foco a solução do problema e não há peculiaridades dos sistemas *Big Data*.

Por fim, para a analise iterativa de dados o sistema Hadoop permite o acoplamento de várias outras ferramentas para esta visualização, entre elas o HBASE. Como o MapReduce é por definição um sistema para processamento de lote esta ferramenta torna a análise de somente uma parte dos dados possível. Além desta ferramenta muitas outras podem ser encontradas desde para captura massiva de dados até a apresentação gráfica das informações.

2.1.5 Extração de dados a partir fontes abertas em rede

Este trabalho se baseia no processamento de dados de ataques a redes e computadores, para extração de informações que possam gerar inteligência sobre ameaças a organizações e países. Para a realização deste processamento é necessário que existam dados a serem processados, por sua vez há algum tempo que estes dados são um ativo organizacional. Assim, eles não são disponibilizados de forma aberta, sendo a maioria deles ou inacessíveis de forma incondicional ou acessíveis somente mediante a pagamento da fonte.

Porém, algumas empresas disponibilizam parte desses dados de forma aberta. Entretanto, tais dados já se encontram pré-processados e apresentam um número de informações reduzidas. Outro desafio é que mesmo os dados se encontrando de forma clara o objetivo não é fornece-los a terceiros, mas sim exibi-los em páginas dinâmicas principalmente como mapas de ataques. Dessa forma sistemas intermediários para captura ou extração

destes dados devem ser desenvolvidas para que estes sejam internalizados no sistema para posterior processamento.

A escolha das fontes se baseou na facilidade de obtenção dos dados e no volume de dados enviados pelas empresas. Duas companhias se destacaram nestes dois quesitos a *Norse Corporation* e a *Looking Glass Cyber*.

2.1.5.1 *NorseCorp*

A *NorseCorp* é uma empresa de Cibersegurança especializada em entregar inteligencia contra ataques de rede, brechas de segurança e ameaças diversas. Ela oferece atualizações contínuas do estado de rede através de vários sensores espalhados pelo mundo.

A empresa reside na cidade de San Mateo na Califórnia. A sua principal solução de segurança é chamada de *IPViking*, ele é um programa responsável por monitorar ativamente as aplicações e máquinas de seus clientes listando os ataques que estão ocorrendo atualmente. Dessa forma é possível identificar a fonte e tipo da ameaça permitindo sanar as vulnerabilidades e reagir de forma proativa a elas.

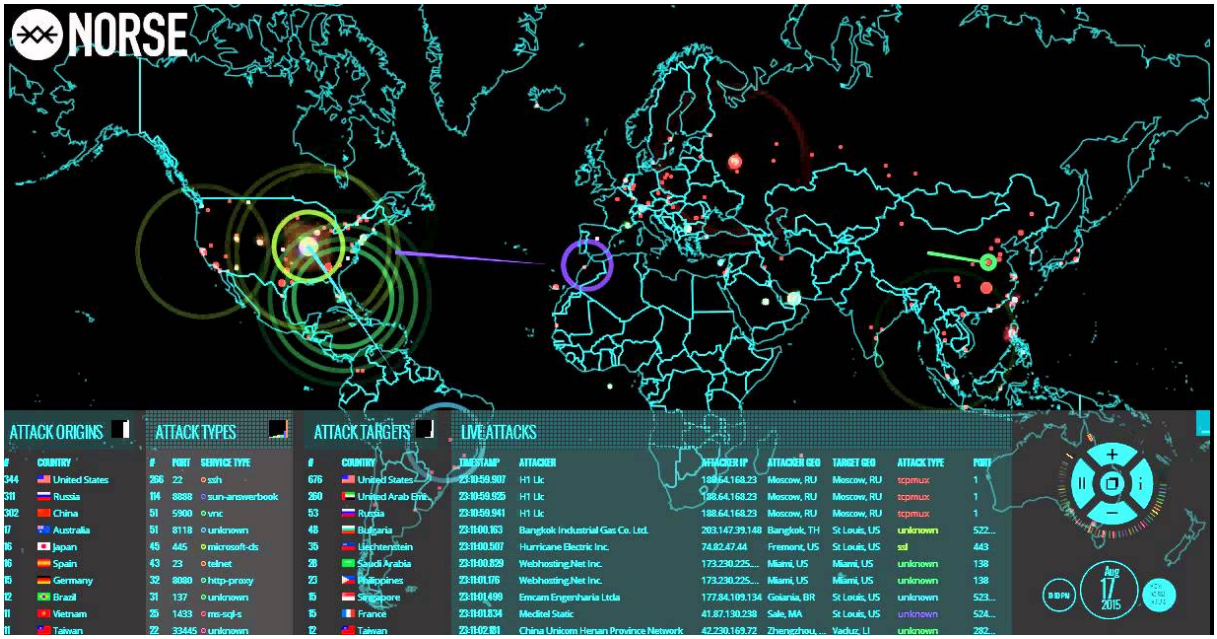


Figura 2.1: Tela principal do mapa interativo da NorseCorp.

Com a finalidade de divulgar a tecnologia utilizada por sua empresa a *NorseCorp* disponibiliza parte destas informações através de uma ferramenta *online* em forma de um mapa interativo. O site pode ser acessado em `http://map.norsecorp.com/`, nele é possível visualizar em tempo real os ataques ocorrendo contra os clientes da empresa. A Figura 2.1 mostra a tela principal da ferramenta.

As informações sobre os ataques são transmitidas dos servidores da empresa até o navegador do cliente utilizando um *WebSocket*, um protocolo definido pela RFC 6455 [37] e um dos principais recursos que surgiram no HTML5 [38].

2.1.5.2 *Looking Glass Cyber*

Outra empresa que expõe tais dados é a *Looking Glass Cyber*, empresa com sede no Condado de *Arlington*, no estado da Virgínia dos Estados Unidos da América. Ela foi fundada em 2009 e tem como especialidade a atuação no mercado de cibersegurança e inteligência contra ameaças.

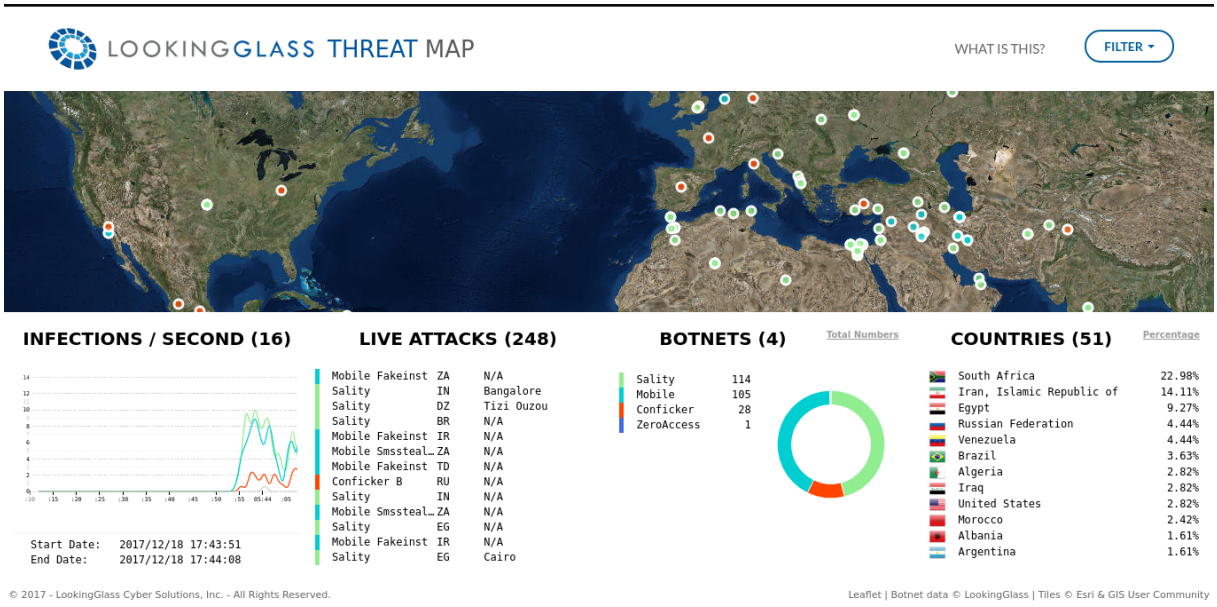


Figura 2.2: Tela principal do mapa interativo da Looking Glass Cyber.

Sua ferramenta de divulgação é semelhante à da *NorseCorp* e está disponível no link `https://map.lookingglasscyber.com/`. Nela um mapa (Figura 2.2), também interativo, é exibido contendo informações sobre ataques à infraestrutura de seus clientes. Os dados também são pré-processados. Uma das informações, em que difere do mapa da outra empresa é a partir de qual *botnet* aquele ataque partiu.

Diversas técnicas para extração ou captura de dados estavam disponíveis, *web crawlers*, programas para download recursivo de páginas, *web spyders*. Mas nenhuma delas era especificamente desenvolvida para lidar com a tecnologia de *WebSocket*, por esse motivo um programa específico para capturar esses dados teve de ser desenvolvido. Os detalhes de implementação e o funcionamento do programa são detalhados no Capítulo 3.

Como o mapa anterior, os ataques são transmitidos dos servidores da empresa até o navegador do cliente utilizando também a tecnologia de *WebSocket*. Esta característica foi levada em consideração para a escolha dessa fonte de dados.

2.1.6 HTTP e *WebSocket*

2.1.6.1 HTTP

O *Hypertext Transfer Protocol* (HTTP) é um protocolo de comunicação utilizado para transmissão de informações de hipermídia, sendo atualmente a base para a *World Wide Web*. O HTTP realiza a transferência de hipertextos, que são textos estruturados com ligações lógicas entre os nós, essas ligações são chamadas de *hiperlinks*. Se encontra atualmente na versão 1.1 tendo sua nova versão já especificada e em aplicação no mercado.

Definido pela RFC 2616 [39], funciona como um protocolo de requisição-resposta, atuando com o modelo arquitetural cliente-servidor. Suas mensagens possuem dois campos: o cabeçalho e o corpo da mensagem.

O cabeçalho é obrigatório sendo utilizado para transmitir informações adicionais entre o cliente e o servidor, e inicia sua especificação logo apos a definição do método. Os campos que podem compor este cabeçalho estão descritos na RFC 2616 (na página 31) e são seguidos por dois pontos (:) e seu valor.

O corpo por outro lado é opcional e não existem grandes definições sobre o seu conteúdo, porém no cabeçalho podem ser definidas suas características, como por exemplo o tipo de dado que está sendo transportados e o seu tamanho.

A mensagem de requisição é enviada pelo cliente e composta pelos campos que podem ser vistos na Figura 2.3a. São eles:

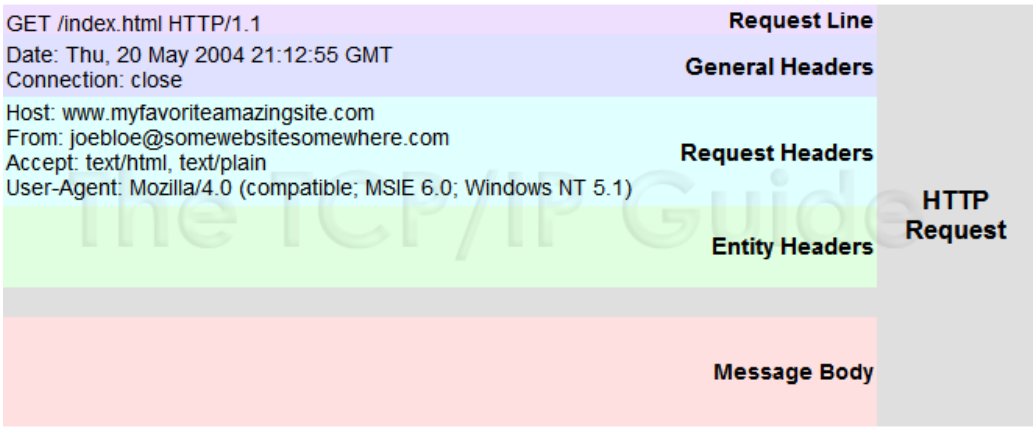
Request Line - Linha que deve iniciar toda mensagem de requisição no HTTP, se a requisição não tiver uma *Request-Line* válida a mensagem é recusada. Essa linha inicial começa com o método HTTP daquela requisição seguido da *URI* da requisição e a versão HTTP utilizada

General Headers - Opções de cabeçalho com propósito geral, podendo ser aplicados tanto a requisições quanto a respostas. Os campos disponíveis para esta seção podem ser encontrados na página 35 da RFC 2616.

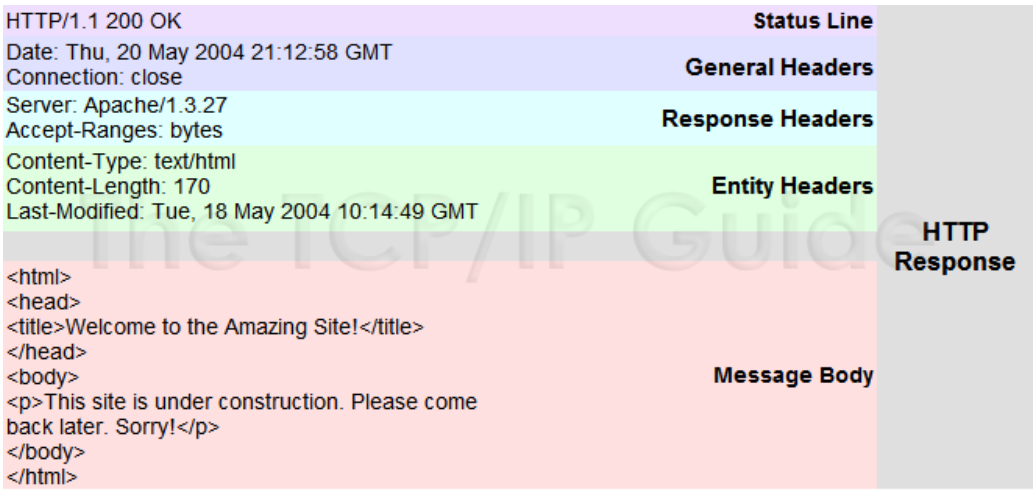
Request Headers - Opções de cabeçalho utilizadas pelo cliente para transmitir informações adicionais da requisição ou sobre o cliente ao servidor. Os seus campos agem como modificadores da requisição.

Entity Headers - Porção opcional que define meta informações sobre o corpo da mensagem quando presente, ou sobre o recurso identificado na mensagem quando o corpo da mensagem não estiver presente.

Message Body - O corpo da mensagem quando presente é utilizado para carregar a mensagem associada com a mensagem de requisição ou resposta. Ela é separada do restante dos cabeçalhos por um linha em branco que é obrigatória para que se possa identificar corretamente onde termina o cabeçalho e começa o corpo.



(a) Exemplo de mensagem de requisição do protocolo HTTP



(b) Exemplo de mensagem de resposta do protocolo HTTP

Figura 2.3: Exemplo de mensagens do protocolo HTTP (Fonte: [40])

A mensagem de resposta é muito semelhante a mensagem de requisição, ela é composta pelos campos podem ser vistos na Figura 2.3b, são eles:

Status Line - Linha que inicia a mensagem de resposta no protocolo. Ela é composta pela versão do protocolo utilizada seguida por um código numérico de estado e seu texto explicativo associado.

General Headers - Opções de cabeçalho com propósito geral, com o mesmo propósito apresentado na requisição e podendo conter os mesmos campos.

Response Headers - Opções de cabeçalho utilizadas pelo servidor para transmitir informações adicionais da resposta ao cliente. Esse cabeçalho fornece informações sobre o servidor e sobre futuros acessos aos recursos hospedados nele.

Entity Headers - Possui o mesmo propósito apresentado na mensagem de requisição.

Message Body - O corpo da mensagem quando presente é utilizado para carregar a mensagem associada com a mensagem de requisição ou resposta. Ela é separada do restante dos cabeçalhos por um linha em branco que é obrigatória para que se possa identificar corretamente onde termina o cabeçalho e começa o corpo.

Em sua versão 1.0, o protocolo HTTP não armazenava informações sobre as conexões anteriormente estabelecidas, sendo desta forma sem estado *(stateless)*. Com essa abordagem cada nova requisição deveria instanciar uma nova conexão, como o protocolo da camada de transporte utilizada é o TCP, estas novas conexões consumiam grandes recursos de processamento e memória do servidor além de largura de banda.

Com o objetivo de contornar este problema, a versão 1.1 do o HTTP trouxe definições sobre conexões persistentes, que tem como objetivo iniciar uma conexão por onde várias requisições e respostas irão ser transmitidas, dessa forma economizando recursos pois só uma conexão TCP é aberta para tráfego de inúmeras mensagens. A conexão é fechada de forma explícita, quando um dos envolvidos não tem mais mensagens a enviar, ou se estiver ociosa por um período determinado de tempo.

As mensagens de requisição HTTP tem em sua primeira linha a definição do método utilizado por aquela mensagem, o HTTP define oito métodos para acesso de recursos. O método indica a ação que o servidor deve realizar sobre o *Uniform Resource Identifier* (URI) fornecido. Os oito métodos são:

GET - Indica para o servidor que ele deve recuperar a informação indicada pelo *Request–URI*, sendo este recurso uma página *HyperText Markup Language* (HTML), um *script*, uma imagem, ou qualquer outro recurso. Esse é o método HTTP mais utilizado.

POST - Utilizado para enviar dados para serem processados pelo servidor, o tipo do processamento é determinado pelo servidor e usualmente é indicado pelo *Request-URI*.

HEAD - É idêntico ao método GET exceto que o servidor não retorna um corpo de mensagem na resposta. Tem o objetivo de transmitir somente as metainformações sobre o recurso que se deseja acessar.

PUT - Envia uma entidade para ser armazenada no servidor, a entidade é identificada pelo *Request-URI*.

DELETE - Exclui uma determinada entidade do servidor, a entidade a ser excluída é identificada pelo *Request-URI*.

TRACE - Invoca um método remoto de eco no servidor. Permite ao cliente visualizar o que está sendo recebido pelo servidor.

OPTIONS - Método para recuperar informações sobre as opções disponíveis para acesso a um determinado recurso, identificado pelo *Request-URI*.

CONNECT - Método reservado para uso com *proxy* para estabelecimento por exemplo de um túnel seguro.

Na mensagem de resposta em sua primeira linha, *Status Line*, o servidor informa a versão e um código numérico de estado. Esses códigos de estado são definidos na especificação do HTTP, eles são inúmeros sendo eles agrupados em classes de estado para uma mais fácil identificação. As classes são:

1xx - Informacional - utilizada para informar o cliente sobre o estado de sua requisição, o cliente deve estar preparado para receber esta classe de estado.

2xx - Sucesso - Utilizada para informar que a requisição enviada teve sua execução bem sucedida.

3xx - Redirecionamento - Informa que o cliente deve executar outra ação para que a requisição possa ser inteiramente concluída.

4xx - Erro no cliente - Informa que a requisição realizada pelo cliente contêm um erro e não pôde ser executada.

5xx - Erro no servidor - Informa que mesmo a requisição sendo válida ocorreu um erro no servidor.

Com esses códigos de estado, o cliente pode ter informações pertinentes sobre as requisições realizadas, como por exemplo, se a reposta contêm o código de estado 200 isto indica que a requisição foi recebida e retornada com sucesso. Se a mensagem for 404 isto indica que o servidor não pôde encontrar o recurso indicado pela *Request-URI*.

2.1.6.2 Tecnologia *WebSocket*

Este protocolo foi desenvolvido como solução para criação de páginas extremamente interativas, que tem suas informações atualizadas em tempo real. Quando estas utilizavam o protocolo HTTP puro existia uma grande sobrecarga sobre o servidor devido a necessidade de se gerenciar conexões que não foram desenvolvidas para aquele propósito.

Com esta tecnologia, agora é possível de forma simples a criação de ambientes com alto dinamismo e alta taxa de troca de informações de forma simples e abstraindo do desenvolvedor as dificuldades e especificidades do gerenciamento de conexões. Seu uso com HTML5 vem criando uma nova geração de páginas *web*.

A tecnologia de *WebSocket* é especificada pela RFC 6455 [37], esse protocolo permite que sejam estabelecidas duas vias de comunicação entre o cliente e o servidor, assim transmissões *full-duplex* podem ocorrer.

O protocolo é composto por duas partes, o aperto de mão (*handshake*) e a transferência de dados. Sendo esse um protocolo desenvolvido para uma total compatibilidade com a infraestrutura existente, ele inicia sua "vida" como uma conexão HTTP comum. Assim, o *handshake* do protocolo acontece utilizando uma requisição com o método GET, sua estrutura é exibida no código 2.1:

```
1 GET ws://echo.websocket.org/?encoding=text HTTP/1.1
2 Origin: http://websocket.org
3 Cookie: __utma=99as
4 Connection: Upgrade
5 Host: echo.websocket.org
6 Sec-WebSocket-Key: uRovscZjNol/umbTt5uKmw==
7 Upgrade: websocket
8 Sec-WebSocket-Version: 13
```

Código 2.1: Mensagem do tipo GET do protocolo *WebSocket*

A primeira linha da requisição trás a descrição do método, a URI que define o protocolo *WebSocket* e a versão do protocolo HTTP. O campo `Connection:` se refere à solicitação para abertura de uma conexão do tipo *WebSocket*, o campo `Sec-WebSocket-Key:` é utilizado pelo servidor para o estabelecer a conexão e compor a reposta que será enviada ao cliente.

Se o servidor entender a mensagem e concordar em estabelecer a conexão, a mudança de protocolos ocorre quando o cabeçalho de atualização é enviado pelo servidor. O cabeçalho para atualização pode ser visto no código 2.2:

```
1 HTTP/1.1 101 Switching Protocols
2 Date: Fri, 10 Feb 2012 17:38:18 GMT
3 Connection: Upgrade
4 Server: Kaazing Gateway
```

```
5 Upgrade: WebSocket
6 Access-Control-Allow-Origin: http://websocket.org
7 Access-Control-Allow-Credentials: true
8 Sec-WebSocket-Accept: rLHCkw/SKsO9GAH/ZSFhBATDKrU=
9 Access-Control-Allow-Headers: content-type
```

Código 2.2: Mensagem de resposta do protocolo *WebSocket*

A reposta do servidor contem o campo `Sec-WebSocket-Accept:` contendo a chave de confirmação que foi gerada a partir da chave enviada durante a requisição. No momento do recebimento dessa mensagem o protocolo HTTP é finalizado e uma conexão *WebSocket* assume seu lugar utilizando a mesma conexão TCP estabelecida anteriormente. O protocolo *WebSocket* utiliza as mesmas portas reservadas ao protocolo HTTP, 80 e 443.

2.2 K-Means

O ser humano é um animal racional, sempre buscando compreender o espaço que o rodeia. Para desenvolver este entendimento ele sempre buscou técnicas para facilitar seu aprendizado, dividir as áreas por grupos, escrevendo livros, o próprio desenvolvimento do método científico teve por objetivo definir uma sequencia de passos que pudesse ordenar o estudo científico.

Observar e reconhecer padrões na natureza sempre foi uma técnica muito utilizada, padrões são vistos nas estrelas, na rotação em torno do sol, nas linguagens e sociedades. A organização e classificação das informações obtidas durante as observações compõe parte importante para o reconhecimento dos padrões.

A análise de dados pode ser dividia em dois tipos [41], exploratória ou descritiva e confirmatória ou de inferência. Na primeira o observador não tem nenhum modelo ou hipótese mas busca entender as características gerais dos dados, na segunda o observador deseja confirmar a validade de seu modelo.

Para o reconhecimento de padrões os modelos preditivos são utilizados, dado um conjunto de dados de treinamento gostaria de se prever o comportamento de um outro conjunto de dados de teste. Duas abordagens para resolver este problema estão disponíveis, o aprendizado supervisionado (classificação) onde os dados de treinamento já estão classificados e o não supervisionado (clusterização) onde os dados não estão classificados [42].

Segundo Anil K. Jain [43], o objetivo da clusterização é descobrir grupos naturais em um conjunto de padrões, pontos ou objetos. Essa técnica é utilizada para iniciar a análise em um conjunto de dados sobre os quais há pouca informação, dessa forma após

a clusterização pode ser mais simples formular hipóteses, e foi por este motivo que ele foi utilizado neste trabalho.

O algoritmo mais simples para particionamento de dados é o *K-Means*. Esse algoritmo não foi descoberto por uma única pessoa mas sim por diferentes cientistas independentes, Steinhaus (1956) [44], Lloyd (proposto em 1957, publicado em 1982) [45], Ball and Hall (1965) [46], e MacQueen (1967) [47]. Ainda há suspeitas de que ele tenha sido proposto primeiramente há mais de 50 anos, sendo ele um dos algoritmos mais utilizados para clusterização dada sua facilidade de implementação, simplicidade, eficiência, e sucesso nos resultados.

A ideia central do algoritmo é encontrar partições em que o erro quadrático entre a média empírica de um *cluster* e seus pontos neste *cluster* seja mínima. O erro quadrático é dado pela equação 2.2, onde $X_i^{(j)}$ é um ponto que representa o dado e C_j é o centro do *cluster*.

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| X_i^{(j)} - C_j \right\|^2 \tag{2.2}$$

Minimizar a função definida pela Equação 2.2 é um problema NP-*hard* mesmo para um K=2 [48]. Entretanto, o *K-Means* é um algoritmo guloso, assim ele somente converge a um mínimo local, há estudos recentes que indicam que existe uma grande probabilidade de que o algoritmo possa convergir para um minimo global quando os *clusters* estão bem separados [49].

O algoritmo funciona seguido os seguintes passos:

1. Posicionar os K pontos no espaço vetorial onde estão representados os dados. Esses pontos representam os centros iniciais dos grupos

2. Calcular a distância de cada ponto aos dois centros e definir qual distância é menor. A menor distância determina a qual grupo o ponto pertence

3. Recalcular os centros dos grupos. Para este passo calcula-se o novo centro como sendo o baricentro dos grupos resultantes do segundo passo

4. Repete-se o passo 2 e 3 até que os centros não alterem mais sua posição

Um exemplo de execução pode ser visualizado na Figura 2.4 onde é possível perceber a alterações dos centroides iniciais a medida que a execução do algoritmo evolui. No fim é possível ver neste exemplo a divisão entre os dois grupos, um identificado pela cor vermelha e o outro identificado pela cor azul.

Este método é bastante sensível a escolha dos centros iniciais. Dependendo da escolha pode-se encontrar centros diferentes sobre um mesmo grupo de dados. Por este motivo,

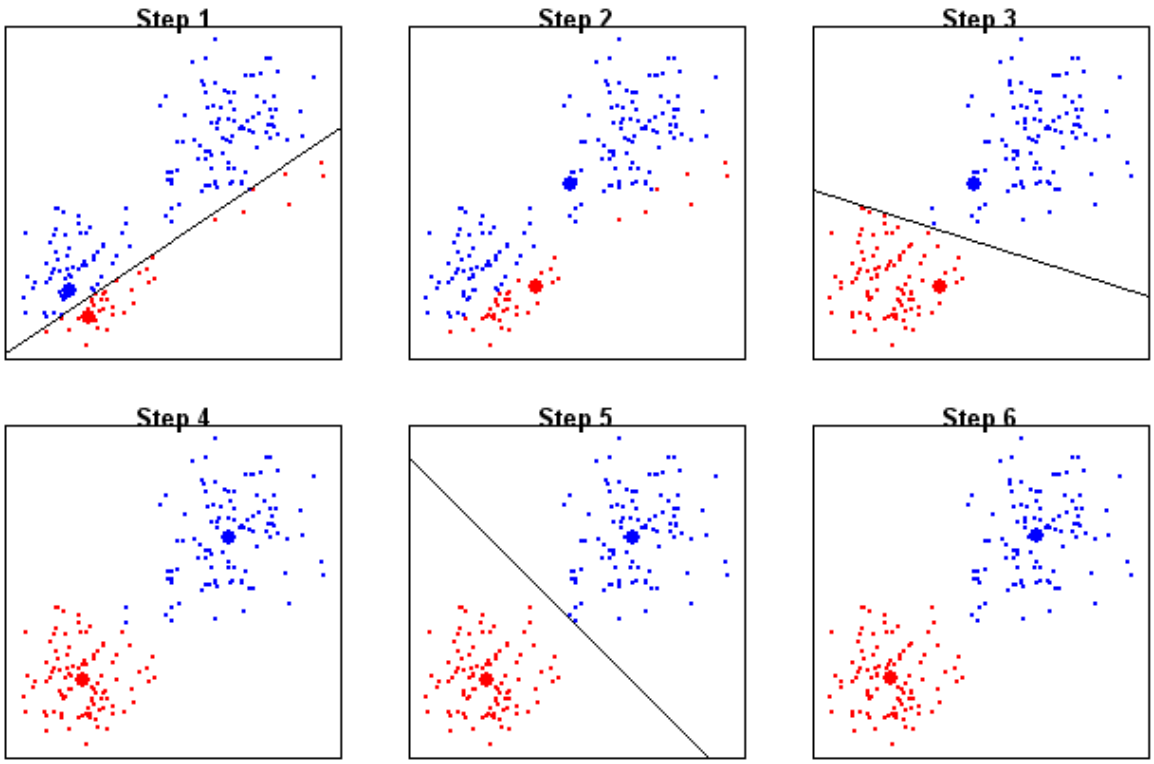


Figura 2.4: Exemplo execução K-Means.

é recomendado que os centros iniciais estejam distantes e que se execute o algoritmo múltiplas vezes para que se reduza esse efeito.

2.3 SciPy

SciPy é um ecossistema composto por uma coleção de iniciativa de *software* aberto para computação científica na linguagem de programação *Python*. Ela conta com módulos para otimização, álgebra linear, integração, interpolação, funções espaciais, transformada rápida de Fourier, processamento de sinais e imagem além de diversas outras ferramentas para cálculo científico e de engenharia [50].

2.3.1 NumPy

É o principal pacote para computação científica, sua principal função é prover uma forma de encapsular os dados na forma de um poderoso objeto vetor multidimensional. Este objeto geralmente é utilizado por todas as outras ferramentas da suíte *SciPy*, esta biblioteca também oferece outras funções: (i) funções de difusão sofisticadas, (ii) ferramentas para integração com as linguagens C/C++ e Fortran, (iii) funções de álgebra linear, transformada de Fourier e número aleatórios [51]. As principais funções utilizadas dessa biblioteca foram: *concatenate*, *array*, *vstack* e *arrange*. Uma breve descrição do funcionamento de cada uma destas funções é fornecida no Código 2.3.

```
1  concatenate((a1,a2,...), axis=0)
2  Concatena uma sequencia de vetores(arrays) ao longo de um eixo existente.
3
4  Parâmetros:
5        a1,a2,...: sequencia de vetores com o mesmo formato
6        axis: int, opcional - eixo ao longo do qual os vetores serão unidos
7  Retorno:
8        res: ndarray - vetor resultante da concatenação
9
10 ************************************************************************
11
12 array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)
13 Cria um vetor(array)
14
15 Parâmetros:
16       object: qualquer objeto que represente um vetor(array)
17       dtype: data-type, opcional - tipo de dado desejado para o vetor(array
       )
18       de saída
19             copy: bool, opcional - define se o object deve ou não ser
```

```
20        copiado
21              order: {'K', 'A', 'C', 'F'} - opcional, define a ordem de
22              saída do vetor(array)
23              ndmin : int, opcional  - define o número minimo de dimensões
24              do vetor de saída
25  Retorno:
26        res: ndarray - vetor resultante seguindo as especificações dos
       argumentos.
27
28  ****************************************************************
29
30  vstack(tup)
31  Empilha os ndarrays de entrada na vertical criando um novo ndarray de saída
32
33  Parâmetros:
34        tup: sequencia de ndarrays a ser empilhada
35  Retorno:
36        stacked: ndarray - vetor resultante dos ndarrays empilhados.
37
38  ****************************************************************
39
40  arange([start, ]stop, [step, ]dtype=None)
41  Retorna um novo ndarray com valores espaçados que são determinado por um
42  intervalo
43
44  Parâmetros:
45        start: number, opcional - início do intervalo incluindo esta posição
46              stop: number - fim do intervalo excluindo esta posição
47              step: number, opcional - intervalo entre os valores, distância
48              entre dois valores adjacentes
49              dtype: dtype - tipo de dado desejado para o vetor(array)
50              de saída
51  Retorno:
52        arrange: ndarray - vetor resultante dos valores espaçados.
```

Código 2.3: Principais funções utilizadas da biblioteca NumPy

2.3.2 SciPy *library*

Sendo o núcleo do ecossistema *SciPy*, nessa biblioteca estão a maioria dos sub-pacotes necessários para realização de computação científica, entre eles o pacote *cluster* que executa todas as operações relacionadas a clusterização dos dados. As duas funções desse pacote utilizadas nesta monografia são a *kmeans* e a *vq*.

A função *kmeans* tem como objetivo definir os *clusters* baseados nos parâmetros retornando um *codebook* e uma distorção. O *codebook* é uma matriz que representa os centroides dos *clusters*. Com esses centros em mãos a função *vq* é aplicada sobre a matriz de ponto para determinar para cada um dos ponto a qual *cluster* ele pertence. Dessa forma é possível construir os gráficos que podem ser vistos no Capítulo 3 em sua seção 3.1.

2.3.3 Matplotlib

Por último a biblioteca *Matplotlib* é responsável pela geração dos gráficos em 2 e 3 dimensões, produzindo figuras com qualidade e variedade de formas além da interatividade. Essa biblioteca torna fácil a geração de gráficos, histogramas, gráficos de barra, entre outros.

Também foi utilizada uma extensão dessa biblioteca para permitir a projeção de coordenadas sobre um mapa do planeta Terra, o nome desta extensão é *Basemap Toolkit*. Com o uso combinado dessas ferramentas foi possível construir gráficos completos com os dados gerados pelas bibliotecas anteriores, facilitando o desenvolvimento e permitindo que o foco permanecesse na solução dos problemas.

2.4 Cloudera Manager

O *Cloudera Manager* é um dos produtos desenvolvidos pela *Cloudera Inc.* [52], uma empresa americana com sede em Palo Alto, Califórnia. Fundada em 2008 por três engenheiros e um executivo aposentado todos com experiência em outras grande companhias, Christophe Bisciglia (Google), Amr Awadallah (Yahoo), Jeff Hammerbacher (Facebook) e Mike Olson (Oracle). Logo em 2009 o engenheiro Doug Cutting, co-criador do Hadoop, ingressou na empresa como Arquiteto Chefe onde permanece até hoje [53].

A empresa vem desenvolvendo outros produtos e recebendo vários aportes, seu primeiro produto o *Cloudera Manager* é uma ferramenta que busca facilitar a gerência e configuração de sistemas para processamento de *Big Data*, em especial o Hadoop.

Gerenciar sistemas distribuídos não é uma tarefa simples, várias atividades têm de ser desenvolvidas periodicamente. Também é constante a demanda por monitorar os serviços, integrar novas máquinas, remover máquinas defeituosas, etc. Com diversas ferramentas disponíveis para processamento e análise de dados é impensável que este controle seja feito de forma totalmente manual.

É com esta ideia que a *Cloudera* desenvolveu seu principal produto. Ele permite que seja automatizado a instalação e configuração de novas máquinas ao *cluster*, oferece uma interface para monitorar e registrar as atividades dos sistemas, facilita a solução de

problema e tem facilidade em sua operação, e implementa várias técnicas pra uma alta disponibilidade.

A instalação deste programa permite gerenciar grandes parques de computadores processando milhares de dados de forma simples e centralizada, reduzindo os custos de operação dos sistemas. A Figura 2.5 exibe um exemplo de todos os serviços que devem ser gerenciados em uma pilha baseada no Hadoop, lembrando que nem todos são obrigatórios.

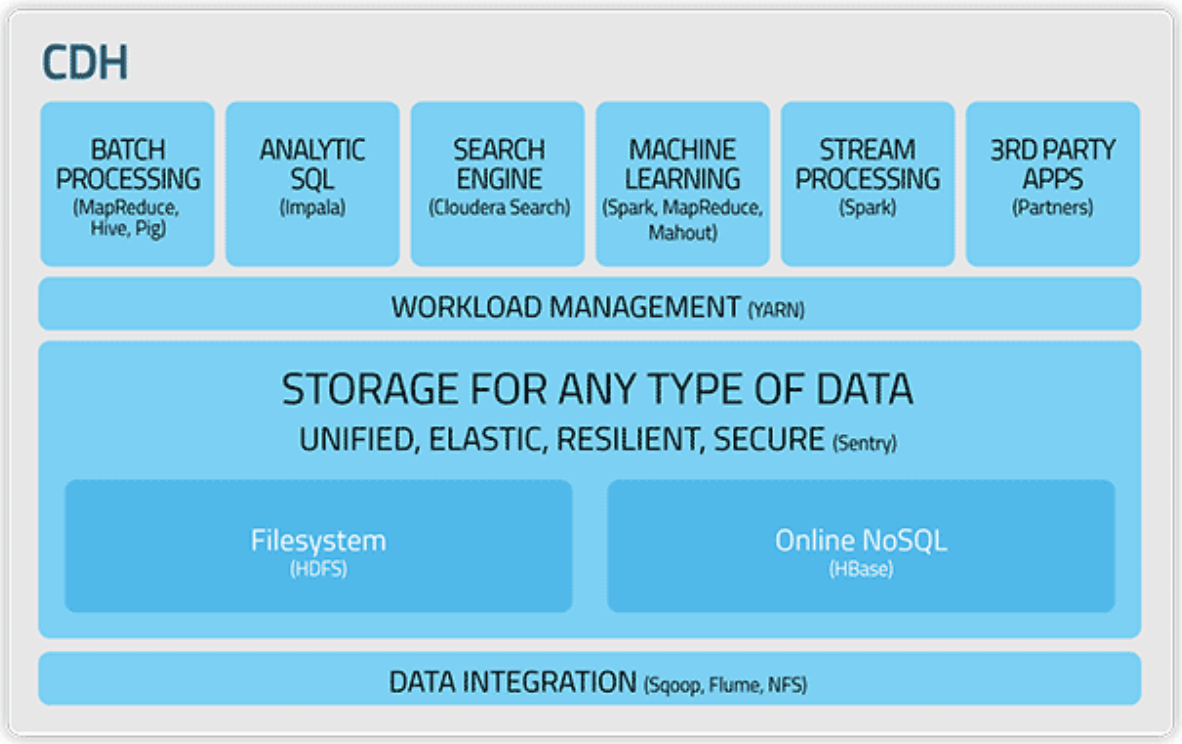


Figura 2.5: Exemplo de ecossitema gerenciado pelo Cloudera Manager.

O *Cloudera Manager* adota a arquitetura cliente/servidor, com um nó central onde o *Cloudera Manager Server* chamado de *SCM Server* é instalado, e a interface de gerenciamento gráfico é hospedada, a Figura 2.6 exibe a página inicial desta interface.

Em cada máquina que pertence ao *cluster* é instalado o *Cloudera Manager Agent* que é responsável por receber os comandos do servidor central e os executar naquele computador. Ainda é sua responsabilidade transmitir informações sobre o estado dos serviços e recursos do equipamento, além de transmitir os registros gerados [54].

O recurso primário de comunicação entre o agente e o servidor é o *heartbeat*, que é uma mensagem enviada pelo agente, a cada 15 segundos, na configuração padrão, ao servidor questionando-o sobre o que ele deveria estar executando. O servidor responde a mensagem com os serviços que deveriam estar em execução, caso o cliente tenha que iniciar um novo processo ele assim o fará.

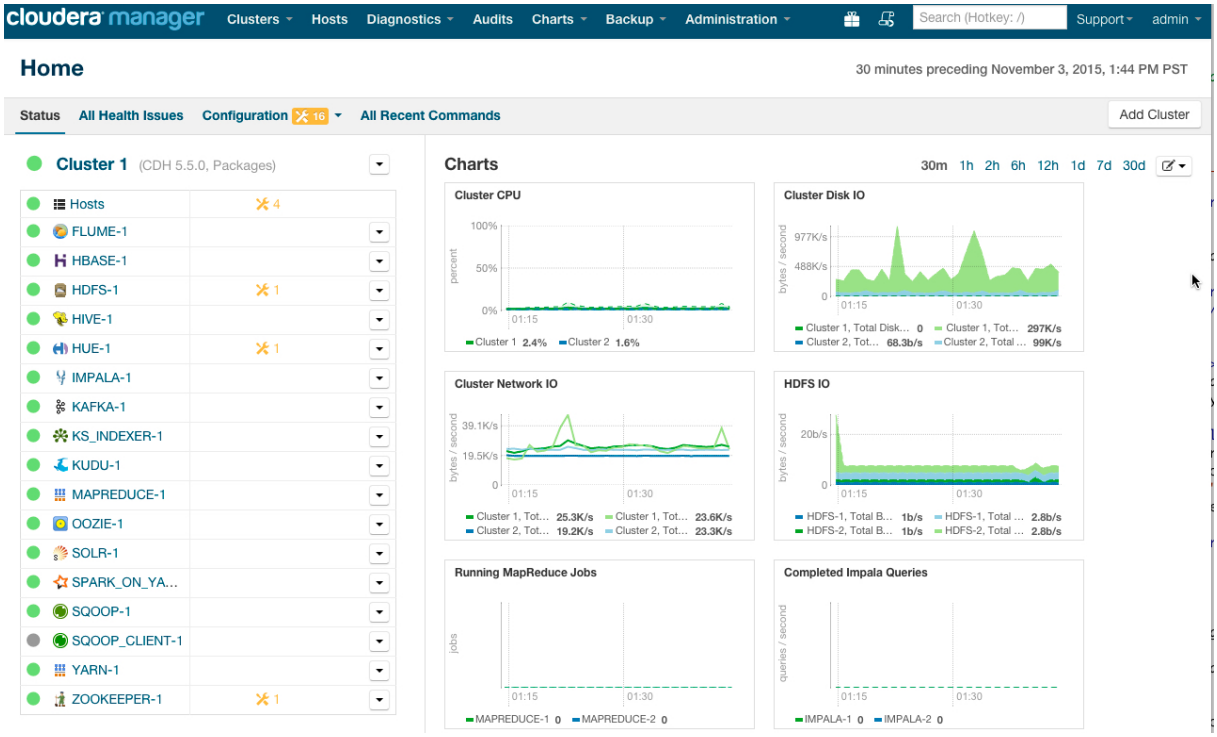


Figura 2.6: Tela inicial da ferramenta de gerenciamento Cloudera Manager.

O servidor sempre mantêm todo o estado atual do *cluster*, as informações são dividas em informações de modelo e de execução. Os modelos definem como o *cluster* deve estar configurado, quantos computadores estão ativos, que serviços cada um deles deve executar, os arquivos de configuração necessários para executar aqueles serviços, dentre outros.

As informações de execução dizem respeito aos processos que estão executando no *cluster* e quais comandos estão sendo atualmente executados. O servidor central também acumula as informações para monitoramento dos serviços possuindo dois processos para isto, o estado de processos e de nós.

Esses processos recolhem uma série de informações baseadas em métricas pré-definidas, e as organiza para serem exibidas no ambiente gráfico. Essas informações oferecem uma boa visão sobre quais nós ou serviços necessitam de mais atenção.

O *Cloudera Manager* foi o primeiro produto da empresa, sua arquitetura é robusta e amplamente difundida no mercado. Contudo, a *Cloudera Inc.* não cessou o desenvolvimento de outros produtos. Atualmente existem outros produtos como o *Cloudera Director* e o *Cloudera Enterprise* que agilizam a instalação de *clusters* na nuvem, sendo que o segundo oferece todo suporte necessário para esse processo.

2.5 Apache Hadoop

O Hadoop é uma plataforma desenvolvida em Java para computação distribuída, amplamente utilizada em *clusters* para processamento de grande volumes de dados. Foi criado por Doug Cutting, o criador do Apache Lucene. Atualmente a *Apache Foundation* é responsável pelo seu desenvolvimento [34].

É um projeto aberto que conta com grandes empresas auxiliando em seu desenvolvimento, tais como Yahoo, IBM, Amazon, Microsoft.

O projeto Hadoop iniciou-se em 2006 derivado do projeto *Nutch* um motor de busca na *web* aberto.

O projeto conta com quatro módulos principais, porém pode ser integrado com diversos outros projetos da *Apache*. Os quatro módulos são:

- *Hadoop Common*
- *Hadoop Distributed File System* (HDFS)
- *Hadoop YARN*
- *Hadoop MapReduce*

O Hadoop Common é o núcleo do sistema, dando suporte e fornecendo uma camada de abstração para que todos os demais módulos possam funcionar corretamente.

2.5.1 HDFS

É um sistema de arquivos portável, distribuído e escalável escrito também em Java, sendo este o sistemas de arquivos padrão do Hadoop. É composto por um *namenode* e vários *datanodes*, o *namenode* é responsável por guardar todas os metadados sobre os arquivos, nome, diretórios, localização de suas partes, onde se encontram as cópias. Se o *namenode* apresentar problemas que o impossibilitem de funcionar corretamente, todo o HDFS deixa de funcionar [55].

Devido a sua criticalidade é recomendado que exista um segundo *namenode* pronto para executar, além é claro de algumas cópias de segurança que devem ser feitas periodicamente.

Os *datanodes* são os nós que realmente armazenam os blocos de dados, estes blocos são transferidos através da rede utilizando um protocolo específico do HDFS. Cada arquivo é divido dentro do *cluster* HDFS em blocos de tamanho padrão, para evitar a perda de informação o HDFS replica cada bloco uma quantidade definida de vezes, três por padrão. Assim cada arquivo possui três cópias distribuídas ao longo do *cluster*. Essa

rotina também é executada para reduzir o tempo de acesso ao arquivo, já que agora ele pode ser feito de forma paralela.

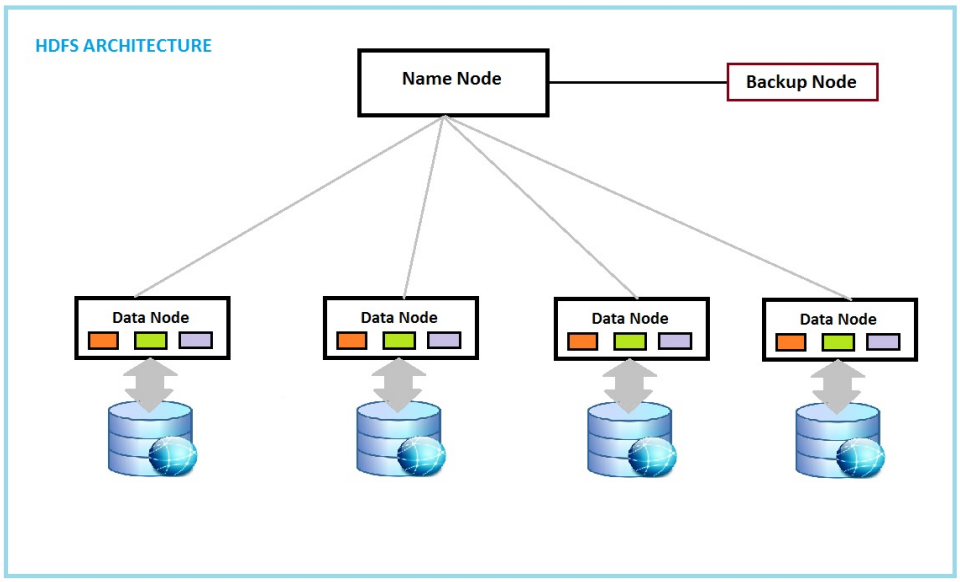


Figura 2.7: Arquitetura do sistema de arquivos HDFS (Fonte: [56]).

A Figura 2.7 ilustra a arquitetura do HDFS, é possível perceber na figura que cada *datanode* possui a posse de três blocos e cada bloco é replicado com fator de três, já que para cada bloco há o bloco original e mais três cópias espalhadas ao longo do *cluster*.

Na Figura 2.7 também é definido um *Backup Node* que é uma cópia ativa do *Namenode*, caso esse venha a falhar o *Backup Node* assume o seu lugar evitando que o sistema deixe de funcionar.

2.5.2 MapReduce

O MapReduce [57] modelo orientado ao processamento dos dados onde duas operações são executadas sobre esses, a função *map* processa uma série de entradas na forma de chave/valor e gera para cada uma dessas zero ou mais saídas no formato chave/valor. A função de *reduce* realiza um agrupamento das saída geradas pela função de *map* podendo gerar um conjunto menor de dados.

A principal vantagem deste modelo é que ele permite que suas operações possam ser executadas sobre conjuntos menores de dados de forma distribuída, sem que ocorra nenhum prejuízo ao resultado. Por exemplo, uma base de 4Gb de dados pode ser executada por vários nós simultaneamente, já que fazendo uso do sistema HDFS os arquivos estão separados em blocos.

Dessa maneira, várias funções *map* podem executar sobre blocos diferentes sem nenhum prejuízo para o processamento dos dados, enviando depois o resultado de sua operação a um nó que esteja executando a função de *reduce*, ou em alguns casos aos nós que estejam executando a função de *reduce*.

A Figura 2.8 exibe um exemplo gráfico da operação MapReduce, nela é possível observar que um conjunto de dados está distribuído em três partes, como se fossem três blocos de um mesmo arquivo. Em seguida cada um dos conjuntos de dados é processado por um nó utilizando a função *map*, a função *shuffle* é utilizada neste exemplo para enviar o resultado da função *map* para o nó correto que está executando a função *reduce*.

O primeiro nó processa os dados representados por retângulos vermelhos, o segundo por retângulos amarelos e o último por retângulos azuis. Caso somente um nó estivesse executando a função de *reduce* a função *shuffle* não seria necessária.

Para concluir a execução, os nós que executam a função de *reduce* a aplicam no conjunto de dados. Em seguida a enviam para um nó central que recebe os resultados parciais e aplica a função de *reduce* novamente, gerando agora o resultado final da operação. Essa operação de *reduce* foi executada em dois passos.

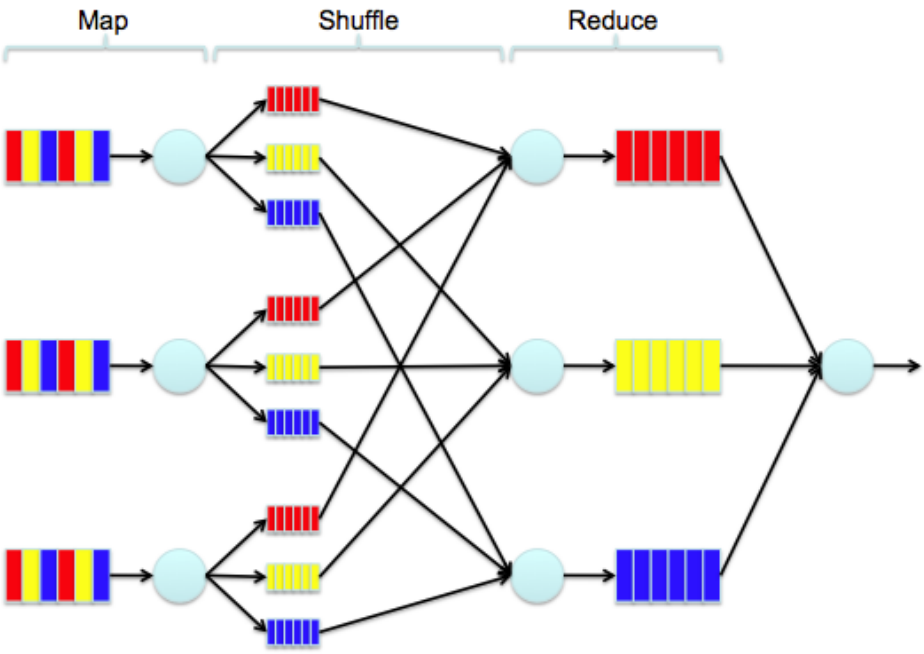


Figura 2.8: Ilustração da operação de MapReduce (Fonte: [58]).

2.5.3 Apache YARN

O *Yet Another Resource Negotiator* (YARN) [59] é o gerenciador de recursos de um *cluster* Hadoop. Foi introduzido na versão 2 para melhorar a implementação do MapReduce, mas em geral suporta bem outros paradigmas de computação distribuída [34].

Esse gerenciador atua como uma camada intermediária entre as aplicações que desejam executar tarefas no *clustesr* e os nós propriamente ditos.

Antes deste gerenciador, as aplicações lidavam diretamente com os recursos do *cluster*, com isto alguns problemas poderiam ocorrer como por exemplo, o desbalanceamento de carga entre os nós do *cluster*. Outro problema, era a replicação de código de controle. Cada aplicação do *cluster* deveria gerenciar suas tarefas para acompanhamento de seu progresso, para isso elas devem monitorar e quando necessário reagendar a execução das atividades não concluídas.

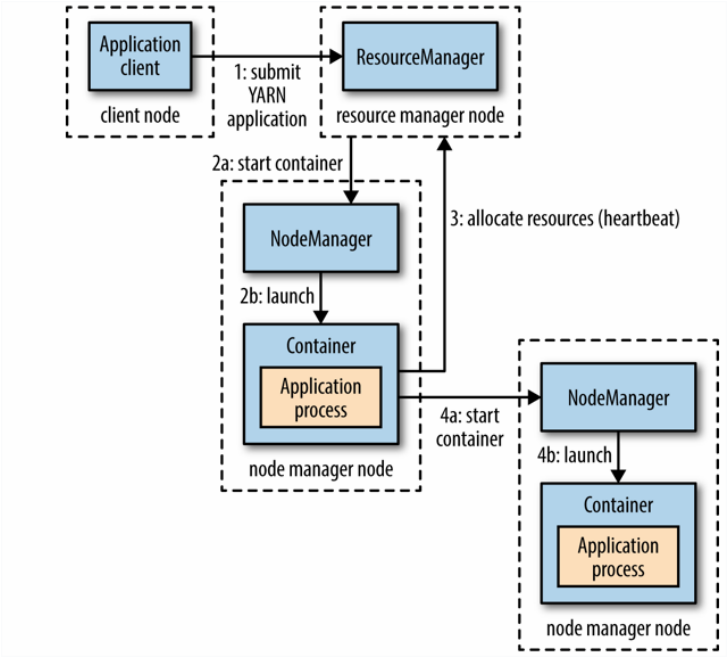


Figura 2.9: Execução de uma tarefa utilizando o Apache YARN (Fonte: [34]).

Para executar uma tarefa utilizando o YARN a aplicação cliente deve solicitar ao YARN que execute a aplicação *master*. Em seguida, o gerenciador procura por um nó que esteja disponível para executar esta aplicação e a inicia dentro de um *container*, essa aplicação pode simplesmente executar seu processamento ou solicitar ao gerenciador mais *containers* a fim de concluir mais rápido o seu processamento. A ilustração desse processo pode ser vista na Figura 2.9.

As vantagens de se utilizar o YARN são: alta escalabilidade podendo chegar até a 10.000 nós executando 100.000 tarefas, alta disponibilidade visto que o gerenciador

utiliza da replicação de estados para evitar sua falha, aumento da utilização do *cluster* e compatibilidade com outros paradigmas.

Sendo assim, a instalação e utilização do YARN é altamente recomendada, o distribuidor *Cloudera* em todos os seus produtos instala e configura por padrão tal ferramenta, melhorando assim o desempenho geral do *cluster*.

2.6 Apache HBase

HBase [60] é uma base de dados distribuída orientada a colunas construída para utilizar os serviços de armazenamento do HDFS. Sendo esta uma aplicação padrão do Hadoop, quando se faz necessário a leitura e escrita aleatória em tempo real de uma grande base de dados [34].

O HBase foi desenvolvido para superar os problemas existentes dos bancos de dados relacionais. Ele aborda o problema da distribuição de dados criando uma estrutura que é facilmente escalável, basta se adicionar novos nós que o espaço total do sistema aumenta. Dessa forma este sistema pode armazenar grande volume de dados de forma distribuída dentro de um *cluster* diferente dos atuais bancos de dados.

O dado no HBase é armazenado em tabelas, cada tabela possui colunas cada coluna pertence a uma família (*column family*). Todas as colunas dessa família de colunas possuem o mesmo prefixo. Cada célula de uma tabela no HBase pode salvar dados versionados, ou seja, inserir um novo dado não exclui o que lá estava anteriormente.

A definição das famílias de colunas deve ser executada durante a construção do esquema da tabela, porém as colunas podem ser adicionadas a qualquer momento à tabela.

Para melhor a velocidade de acesso aos dados, o HBase particiona as tabelas de forma horizontal, e as distribui entre os nós do *cluster* que aqui são chamados de regiões. O sistema tenta sempre manter as regiões com o mesmo número de registros a fim de não sobrecarregar demasiadamente as suas regiões.

O funcionamento do HBase depende do programa ZooKeeper (que será abordado na seção seguinte). Esse programa armazena as informações necessárias para o acesso aos dados das tabelas, media as transações entre regiões e mantem informações sobre elas para que seja possível recuperar uma região que venha a sofrer problemas. A Figura 2.10 ilustra a distribuição dos papeis no HBase.

2.7 Apache ZooKeeper

O Apache ZooKeeper [61] é um coordenador de serviços com alta performasse e alta disponibilidade. Ele permite que processos distribuídos se coordenem entre si através de

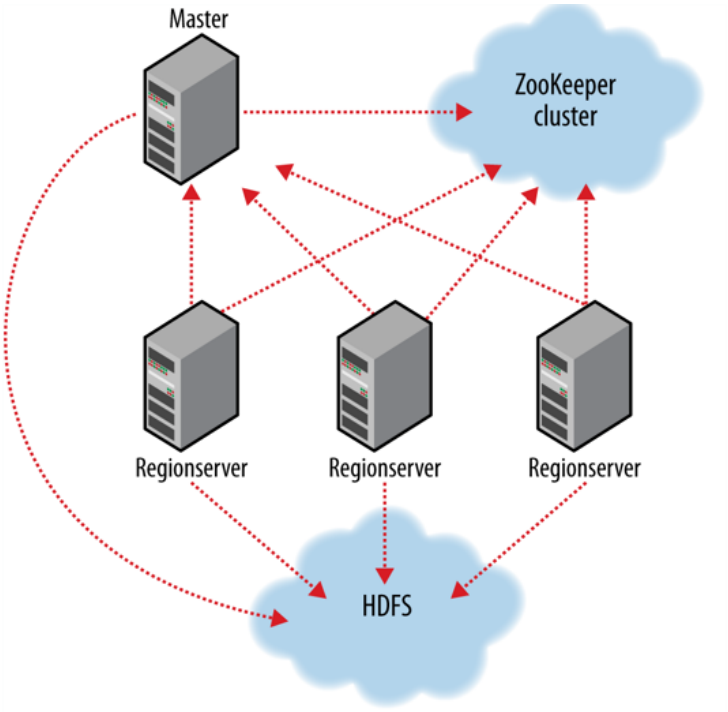


Figura 2.10: Anatomia do funcionamento do programa HBase (Fonte: [34]).

um espaço de nome hierárquico compartilhado de registros de dados (chamamos esses registros de znodes).

Em linhas gerais ele permite uma sincronização confiável entre nós. Para isso os nós desse serviço mantêm uma imagem em memória, da árvore de dados, juntamente com um registro de transações e instantâneos de forma persistente. Ele ainda provê a seus clientes um alto fluxo de dados com baixa latência, alta disponibilidade e também acesso estritamente ordenado aos znodes. O próprio serviço do ZooKeeper é replicado em um conjunto de nós garantindo sempre a execução das operações.

Como os dados são mantidos na memória, o ZooKeeper pode obter altas taxas de transferência com baixa latência. A desvantagem de um banco de dados em memória é que o tamanho do banco de dados que o ZooKeeper pode gerenciar é limitado pela memória. Essa limitação é mais uma razão para manter a quantidade de dados armazenada pequena nesses znodes.

Os servidores que compõem o serviço do ZooKeeper devem conhecer uns aos outros. Enquanto a maioria dos servidores estiver disponível, o serviço do ZooKeeper estará disponível. Os clientes também devem conhecer a lista de servidores, de posse dela os clientes criam uma conexão para o serviço ZooKeeper usando esta lista de servidores.

Os clientes apenas se conectam a um único servidor do ZooKeeper. O cliente mantém uma conexão TCP através da qual envia pedidos, recebe respostas, recebe eventos de

exibição e envia *heartbeats*. Se a conexão TCP para o servidor fechar, o cliente se conectará a um servidor diferente. Quando um cliente se conecta ao serviço ZooKeeper, o primeiro servidor configurará uma sessão para o cliente. Se o cliente precisar se conectar a outro servidor, esta sessão será restabelecida com o novo servidor [62].

Esse serviço é essencial pois a sincronização de estado entre nós é uma característica obrigatória para o funcionamento da maioria das tarefas em um *cluster*. Sendo que, o ZooKeeper oferece total suporte para transmissão destes estados de forma segura e prática.

2.8 Apache Spark

O Apache Spark [63] é uma ferramenta de computação para processamento de grande volume de dados. Ao contrario de outras ferramentas do Hadoop, o Spark não utiliza o MapReduce em suas rotinas. Entretanto, ele possuí várias semelhanças com ele, sendo que o Spark é totalmente integrado ao Hadoop: executa utilizando o YARN e é compatível com o formato de arquivos HDFS [34].

A diferença fundamental do Spark é sua capacidade de manter grandes massas de dados em memória entre as tarefas, permitindo assim que ele tenha um desempenho equivalente, e muitas vezes melhor, que o MapReduce. Dois estilos de aplicação se beneficiam mais da utilização do Spark, algoritmos iterativos e analises interativas.

Outras características permitem que o Spark seja a melhor solução para alguns problemas, como por exemplo: o Spark sempre busca manter a maior quantidade possível de dados em memória, ele retém os resultados intermediários na memória evitando escrevê-los em disco, suporta mais funções do que o MapReduce, otimiza o uso de operadores de grafos arbitrários, executa avaliação sobre demanda nas consultas de *Big Data*, fornece suporte a múltiplas linguagens de programação e ainda oferece um *shell* interativo para desenvolvimento.

A arquitetura do Spark possui três componentes: o armazenamento de dados, a API e a ferramenta de gerenciamento. O Spark funciona com a maioria dos sistemas de arquivo do Hadoop, como o HDFS, HBase. A API é o núcleo do Spark que permite que as instruções sejam executadas. Por fim o gerenciador permite executar as tarefas do Spark de forma local ou em um *cluster*, é papel do gerenciador é fazer a interface entre o programa e o gerenciador de recursos do *cluster*.

O coração do Spark são seus *Resilient Distributed Dataset* (RDD)s, eles são as estruturas de dados utilizadas por ele. Elas tem como características serem particionadas e tolerantes a falhas, os RDDs podem ser criados de diversas maneiras e são imutáveis. Podem-se aplicar transformações ao RDD porém o resultado da transformação será um novo RDD sendo que o original permanece intocado.

Quando essa transformação é executada ela gera um registro, que pode ser utilizado para refazer a transformação em caso de falha, o grafo produzido pelas transformações nas RDDs é chamado de *Lineage Graph*. Ainda existe uma segunda classe de operações, as ações. Quando uma ação é executada ela avalia o RDD e retorna um novo valor, realizando todas as consultas de processamento de dados.

O Spark não é a única biblioteca, ela pertence a um ecossistema que possui outras quatro: Spark Streaming, Spark SQL, Spark MLlib e Spark GraphX. Todas elas executam sobre o núcleo do Spark e compartilham a todas as suas características.

2.9 Apache Flume

Como já mencionado, o Hadoop é um sistema capaz de processar grandes volumes de dados, para que seja possível este processamento os dados devem estar armazenados em seu sistema de arquivos, HDFS. Entretanto, alguns sistemas ainda não tem suas bases de dados internalizadas, mas sim geram estes dados em *stream* e desejam que eles sejam agregados, armazenados e analisados utilizando o Hadoop [34].

O Apache Flume [64] foi desenvolvido para prover a capacidade de ingestão de grandes volumes de dados de forma distribuída e confiável. Como por exemplo realizar a leitura de registros de *logs* de uma aplicação, e.g. um servidor *web*, agregar os arquivos e armazena-los no HDFS. Outros sistemas de arquivos também são suportados pelo Flume, tais como o HBase e o Solr.

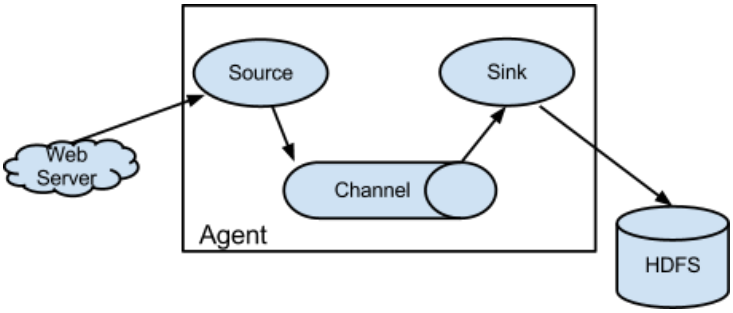


Figura 2.11: Arquitetura básica de funcionamento do serviço Apache Flume (Fonte: [65]).

O Flume é composto por um processo Java que executa as suas funções principais, *sources*, *sinks* e *channels*. O *source* no Flume é o ponto de entrada dos dados, ele tanto pode escutar uma porta esperando por eles ou ativamente ler um determinado arquivo ou diretório. Sempre que um novo dado é inserido o *source* transforma esse em um evento e o envia para o *channel* que o armazena até que ele seja entregue a *sink*. A Figura 2.11 ilustra a arquitetura básica do Flume.

O grande diferencial do Flume é que ele conta com um sistema de transações para garantir que os dados sejam entregues corretamente. Quando o *source* gera um novo evento ele só marcará que esse evento foi armazenado quando receber a confirmação que ele foi inserido no *channel* com sucesso. O mesmo acontece entre o *channel* e a *sink*. Caso ocorra algum problema com o evento e ele não possa ser confirmado como entregue, a transação retorna e o evento fica armazenado aguardando para ser reenviado posteriormente

Várias opções de configuração estão disponíveis para cada uma das funções, por exemplo o *source* pode se conectar a diversas aplicações como: Avro, Twitter, JMS, ou receber dados de uma conexão HTTP [34]. Há também a opção de realizar a leitura de arquivos de *log*, leitura de diretórios, saída padrão e Netcat. Essas opções tornam muito simples utilizar o Flume, em vez de adaptar os dados para que eles possam ser ingeridos corretamente é possível configurá-lo da melhor maneria possível.

As *sinks* e *channels* também possuem opções diversas para sua configuração. As opções para as *channels* são as que mais apresentam diferenças entre si, se a opção de um *file channel* for escolhida os eventos são armazenados em um arquivo. Assim mesmo que o sistema reinicie os eventos ficarão armazenados aguardando para serem enviados, entretanto, caso um *memory channel* seja utilizado os eventos permanecem somente em memória, logo se o sistema reiniciar esses serão perdidos.

Para as *sinks* também existem várias opções, permitindo que os dados sejam ingeridos para diversos sistemas de arquivo como por exemplo, HBase, HDFS, IRC, Solr, arquivos e outros.

As conexões entre as funções apresenta também uma certa diversidade, são permitidos diversos arranjos de forma a adequar o sistema as diversas necessidades do usuário. Utilizar dois *channels* redundantes, duas *sinks*, dois *channels* conectados a mesma *sink*, o que for necessário para se atingir o objetivo. Alguns exemplos de configuração de conexão entre os componentes do Flume podem ser visto na Figura 2.12.

2.10 Apache Kafka

O Apache Kafka [69] é uma plataforma de transmissão distribuída, cujo objetivo é criar um serviço unificado com alta vazão e baixa latência para tratamento de dados em tempo real. Essencialmente ele é uma fila de mensagens amplamente escalável que atua com transações distribuídas [70].

O funcionamento do Kafka pode ser definido com três ações, um produtor envia uma mensagem ao sistema, essa mensagem é recebida e anexada a um tópico e então um consumidor remove essa mensagem e a processa.

Sua capacidade lidar com um grande volume de dados em tempo real é explorada de diversas maneiras, o Apache Flume permite que o Kafka seja utilizado como *source*, *channel* ou *sink*. Desta forma seu uso combinado com esse programa é muito utilizado.

Seu uso como *channel* no Flume produz um *channel* com elevada confiabilidade e disponibilidade, além de apresentar desempenho superior aos *channels* disponíveis no Flume. Lembrando que o Flume só disponibiliza, nativamente, três opções para os *channels*: *file channel*, *memory channel* e JDBC.

Entre as opções somente o *file channel* e o JDBC garantem que as mensagens não serão perdidas caso o sistema falhe, porém estas duas opções introduzem um atraso gigantesco à transmissão das mensagens já que é necessário um processo de escrita em disco. Utilizando o Kafka como *channel* além de garantir que as mensagens não serão perdidas o desempenho em seu processamento permanece alto, o programa garante que mesmo com a falha de um servidor Kafka os dados serão entregues, visto que toda mensagem é replicada entre outros servidores para garantir sua disponibilidade.

Utilizar o Kafka como *source* ou *sink* apresentam iguais vantagens dependendo da especificação dos serviços. Essa opções não serão descritas em detalhes pois não formam utilizadas neste trabalho.

2.11 Apache Solr

Apache Solr [71] é um servidor de buscas de alto desempenho derivado do projeto Apache Lucene, utilizando o Lucene Core como base para seus mecanismos de indexação e busca [72]. O Solr indexa os documentos de forma a permitir buscas textuais a eles. Uma variedade de documentos podem ser indexados entre eles estão: xml, json, csv, pdf, doc, docx, ppt, pptx, xls, xlsx, odt, odp, ods, ott, otp, ots, rtf, htm, html, txt e log.

Grandes empresas vem utilizando essa ferramenta, dentre as principais pode-se citar: Netflix, Instagram, Disney, Apple, CISCO [73]. O sistema é extremamente robusto, escalável e tolerante a falhas contando com uma série de recursos para auxiliar nas buscas.

Os principais recursos do Solr, incluem: busca textual, destaque de parte dos documentos, busca facetada e analítica, análise complexa de documentos, busca geoespacial, permitir a utilização de REST APIs bem como em paralelo com SQL.

Para indexar de forma ótima os dados, o Solr disponibiliza a definição de tipos, campos dinâmicos e campos calculados. A configuração de tipos permite definir o tipo de cada dado dentro do documento e sua precisão. Também permite a definição de analisadores e filtros sobre os tipos, de modo a permitir definir o seu comportamento.

O principal arquivo de configuração do Solr é o *schema.xml* através dele é possível definir um modelo de dados para ser utilizado durante a indexação. Os campos podem

ser obrigatórios, opcionais, dinâmicos ou calculados. Os campos dinâmicos são utilizados quando a forma do documento muda entre um documento e outro, utilizando este artifício é possível definir um padrão em tempo de execução. Os campos calculados são compostos pelo processamento de outros campos, assim campos mais complexos podem ser gerados a partir de outros.

O Solr possui rica interface de administração que permite exportar, importar, realizar buscas, definir modelos e outros. Sua capacidade de indexar e realizar busca em tempo quase real torna essa ferramenta fundamental para análise de subconjunto de dados, oferecendo a capacidade de dividir estes dados sem grandes dificuldades.

2.12 Hadoop User Experience(HUE)

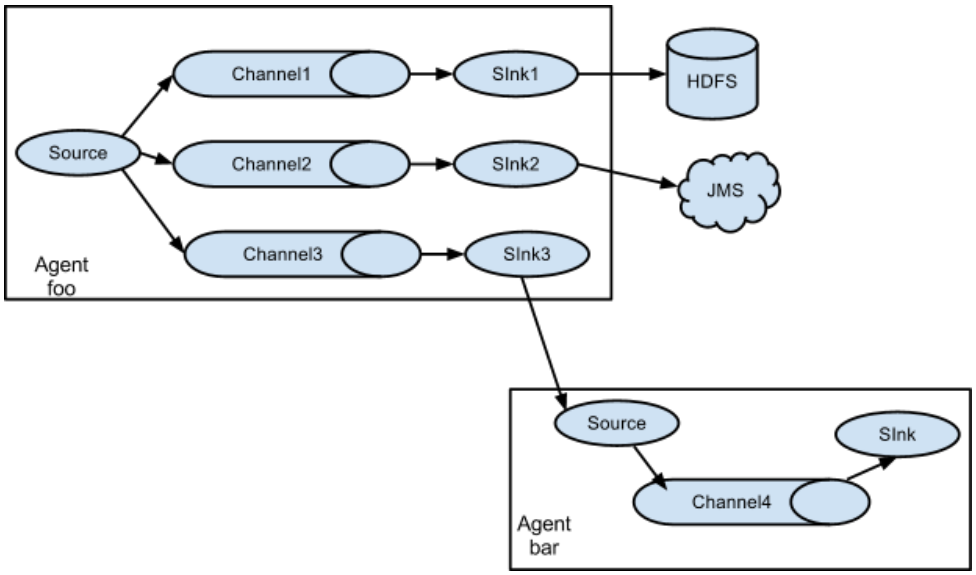
Uma grande variedade de aplicações e serviços estão disponíveis para o Hadoop, todas elas contam com sua própria interface de configuração, visualização, monitoramento. Utilizar várias aplicações para realizar uma análise mais completa e abrangente vem se tornando necessário, visto o aumento dos dados e os benefícios de cada uma das ferramentas.

O *Hadoop User Experience* (HUE) é uma interface *Web* para interação com as diversas aplicações do Hadoop. Como toda aplicação para o Hadoop, ele é distribuído, escalável e tolerante a falhas. Com ele, é possível editar diretamente os dados e configurações das principais aplicações: Hive, Impala, Pig, MapReduce, Spark e qualquer sistema SQL like MySQL, Oracle, SparkSQL, Solr SQL, Phoenix.

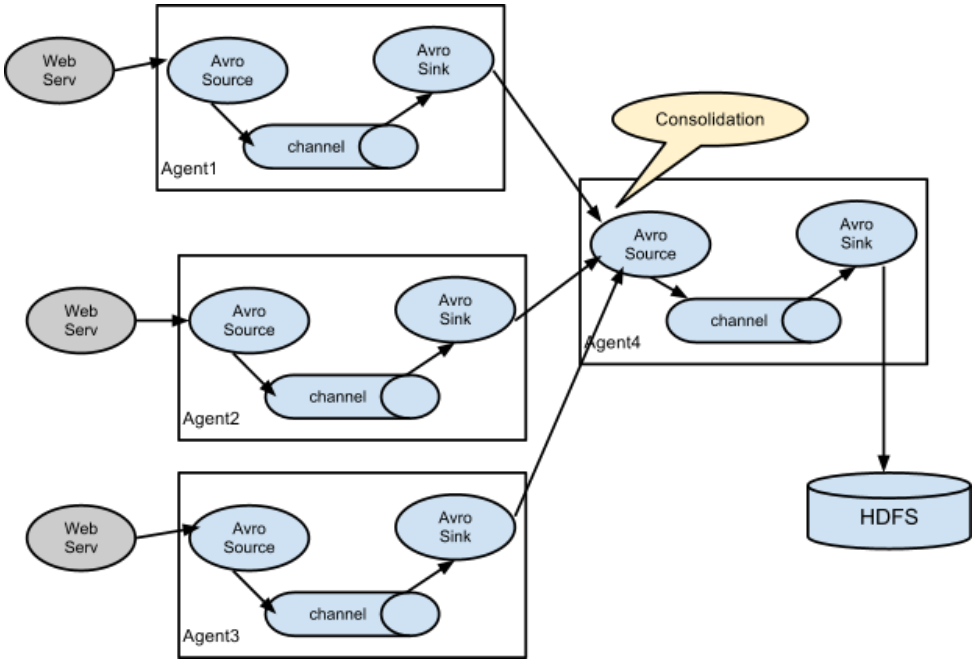
Ele disponibiliza também um painel para visualizar dinamicamente os dados do Solr ou de sistemas SQL. E a possibilidade de agendar e monitorar os trabalhos em execução dos serviços HDFS, arquivos Amazon S3, tabelas SQL, índices, arquivos Git , permissão Sentry, Sqoop. Um exemplo desse painel é ilustrado na Figura 2.13.

Ele foi desenvolvido pela *Cloudera Inc.* porém é uma iniciativa de *software* aberto, estando hoje disponível para instalação pelos principais distribuidores do Hadoop.

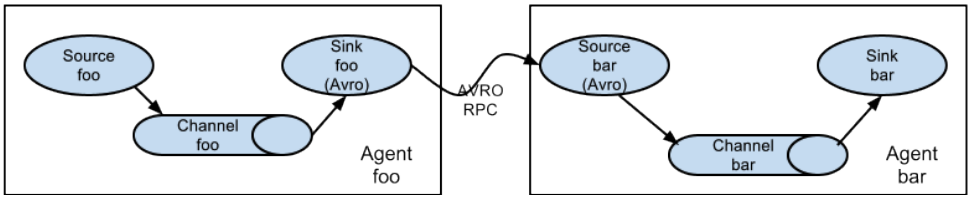
Com esse programa a visualização e edição dos dados alcança outro patamar, além de uma interface limpa e agradável os dados são apresentados de forma a permitir uma análise rápida. Permitindo assim ao profissional que os analisa focar na analise dos dados e não em comandos e peculiaridades de cada uma das aplicações que os estão processando.



(a) Exemplo de conexão de elementos do Flume com múltiplos *channels* e *sinks* (Fonte: [66])

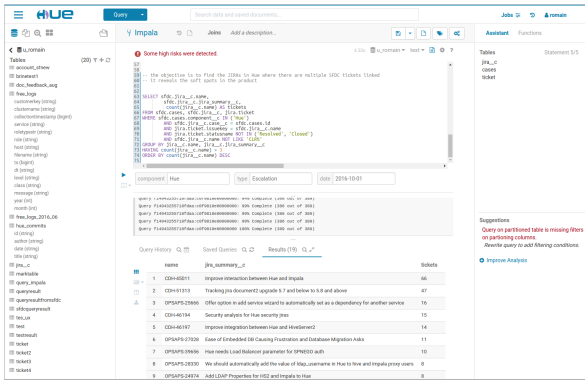


(b) Exemplo de conexão de elementos do Flume com múltiplos agentes (Fonte: [67])

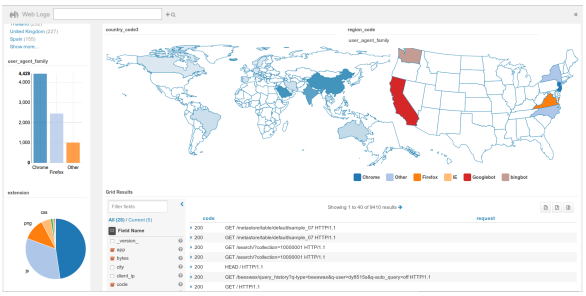


(c) Exemplo de conexão de elementos de agentes do Flume (Fonte: [68])

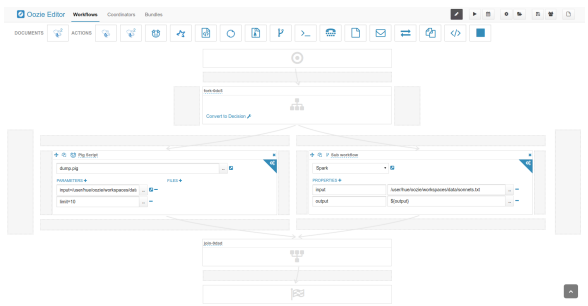
Figura 2.12: Exemplos de conexão de elementos do serviço Flume



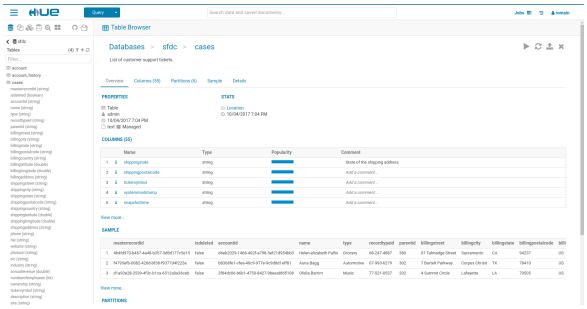
(a) Hue HIVE Editor



(b) Hue Dashboard



(c) Hue Workflows



(d) Hue Table Browser

Figura 2.13: Exemplo de telas da ferramenta HUE (Fonte: [74])

Capítulo 3

Projeto e Implementação

Existem diversas fontes abertas de dados sobre ataques de redes e computadores na internet, essas fontes disponibilizam geralmente um subgrupo de dados que foi obtido pela sua organização como por exemplo porta atacada, IP do atacante, hora do ataque, tipo de vulnerabilidade explorada, dentre outros.

A captura desses dados geralmente é simples, visto que eles se encontram disponíveis para acesso *Web* por meio do protocolo *Hypertext Transfer Protocol* (HTTP) descrito na subseção 2.1.6, dessa forma os dados podem ser obtidos por meio de programas simples.

O projeto consiste em capturar dados sobre ataques através dessas fontes abertas e persistí-los, de forma que, seja possível realizar operações de consulta e análises posteriores. Os métodos de análise utilizados se concentram naqueles específicos para clusterização e extração de informação de forma não-supervisionada. A *posteriori*, o objetivo é integrar diversas bases de dados para construção de conhecimento geral e resiliente sobre ataques em âmbito local e global.

Para tal estrutura, é necessário desenvolver um ambiente que conte com redundância, elasticidade e escalabilidade, visto que o número de dados obtidos é vertiginosamente elevada.

3.1 Primeira Abordagem

Iniciando o desenvolvimento do projeto a primeira abordagem se sustentou na captura de dados da empresa norte-americana *NorseCorp* já descrita na subseção 2.1.5.1 que disponibiliza os dados sobre ataques realizados contra seus clientes em uma página na internet.

O primeiro objetivo era conseguir realizar a captura dos dados e persistí-los em um banco de dados local. Em seguida utilizar o algoritmo de clusterização *K-means*, descrito na seção 2.2, sobre alguns subconjuntos de dados para explorar a possibilidade de

identificação de padrões para enriquecimento os dados armazenados com a geração de conhecimento.

Como diversas incertezas existiam sobre o projeto, a escolha do desenvolvimento de uma aplicação mais simples foi a forma encontrada para reduzir os riscos envolvidos sem grandes custos de desenvolvimento e operacionais, e assim avaliar a viabilidade como um todo.

3.1.1 Arquitetura

A linguagem de programação *Python* foi escolhida para o desenvolvimento tanto da ferramenta de captura quanto para a análise dos dados. Essa escolha residiu no fato de que tal linguagem possui uma curva de aprendizado muito suave, além de possuir diversas bibliotecas em sua estrutura padrão permitindo que se concentre na solução do problema ao invés de resolver problemas pontuais de implementação e adaptações para correto funcionamento dos programas.

O banco de dados selecionado para armazenamento dos dados foi o *MongoDB*. Por se tratar de um banco não relacional detalhes de modelagem de tabelas e relacionamentos poderiam ser eliminados do projeto. Uma outra vantagem é a facilidade de se exportar os dados contidos neles para formatos semi estruturados como por exemplo o formato *JavaScript Object Notation* (JSON), que apresenta como vantagem sua versatilidade e velocidade de processamento. Como última observação esse banco de dados já possui bibliotecas para interação com a linguagem escolhida.

A arquitetura do sistema descrita na Figura 3.1 consiste em um servidor local onde os programas de captura e processamento de dados se encontram em execução. A conexão ao site da *NorseCorp* é feita por meio de um *WebSocket* como referido na subseção 2.1.6. Após o recebimento dos dados pelo programa de captura esse os envia para o servidor de banco de dados para armazenamento. O servidor de banco de dados se encontra na máquina identificada como servidor local.

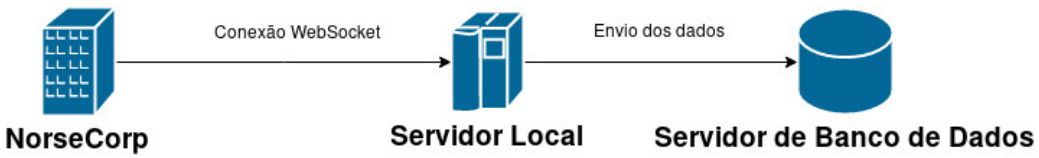


Figura 3.1: Arquitetura proposta para a primeira abordagem.

Essa arquitetura simples permite uma alta vazão na captura de dados da fonte, seu desenvolvimento é simples e confiável.

3.1.2 Captura de Dados

O protocolo para captura dos dados busca simular uma conexão legítima de um navegador para Internet ao site da empresa. Utilizando a biblioteca *websocket-client*, o programa cria uma conexão do tipo *websocket* diretamente com o servidor da empresa. Após estabelecida a conexão por parte do programa só é necessário manter uma espécie de *heartbeat* com o servidor *web* para manter a conexão do tipo *WebSocket* aberta. Grande parte das fontes de dados de ataque utiliza a tecnologia de *WebSocket* para transmissão dos dados. Dessa forma o programa é altamente adaptável e reusável.

Muitas outras fontes de dados sobre ataques podem ser encontradas na internet, como por exemplo:

- *ThreatMap CheckPoint* - `https://threatmap.checkpoint.com/ThreatPortal/livemap.html`
- *Digital Attack Map* - `http://www.digitalattackmap.com/#anim=1&color=0&country=ALL&list=0&time=17487&view=map`
- *Cyber Map Kaspersky* - `https://cybermap.kaspersky.com/`
- *FireEye Cyber Map* - `https://www.fireeye.com/cyber-map/threat-map.html`

Muitas outras organizações e páginas independentes trazem dados dessa natureza.

A captura de dados segue o algoritmo definido pela tecnologia *WebSocket*. Como já explanado, na subseção 2.1.6, nesse protocolo existem quatro tipos de mensagens: *on_message*, *on_error*, *on_close* e *on_open*. O pseudo código abaixo descreve o funcionamento do programa de captura:

```
1  funcao on_message(fd_websocket, message):
2      se timeout:
3          Fecha conexao WebSocket;
4      senao se mensagem recebida igual a ("rid":1) entao:
5          Envia mensagem ('{"event":"#subscribe","data" :
6                  {"channel":"global"},"cid":2}');
7
8      senao se mensagem recebida igual a ("rid":2) entao:
9      Confirma que WebSocket foi aberto;
10     senao se mensagem recebida igual a (#1) entao:
11         Imprime("Ping recebido!");
12         Envia mensagem ("#2");
13     senao:
14         timeout <- 5;
15         mensagem_json <- mensagem;
16         para ataque em message_json:
```

```
17              ataque['timestamp'] <- hora_utc_atual();
18              Insere ataque no banco de dados;
19
20  funcao on_error(fd_websocket, error):
21      Imprime(error);
22
23  funcao on_close(fd_websocket):
24      Fecha conexao com o banco de dados;
25      Imprime("Conexao finalizada");
26
27  funcao on_open(fd_websocket):
28    Envia mensagem ({"event":"#handshake","data":{"authToken":null},"cid":1})
29
30  funcao main():
31    Cabecalho <- {'Accept-Encoding' : 'gzip, deflate, sdch',
32      'Accept-Language':'pt-BR,pt;q=0.8,en-US;q=0.6,en;q=0.4,es;q=0.2',
33      'Cache-Control':'no-cache',
34      'Pragma':'no-cache',
35      'Sec-WebSocket-Extensions':'permessage-deflate; client_max_window_bits'
      ,
36      'User-Agent':'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36'}
37
38      cliente <- Recebe Descritor do Banco de dados
39      banco <- Recebe Descritor da Base de dados
40      attack <- Recebe Descrito da \textit{Collection}
41
42      fd_websocket <-Abre WebSocket("ws://map.norsecorp.com/socketcluster/",
      header = headerc,
43        on_message = on_message,
44        on_error = on_error,
45        on_close = on_close)
46
47      fd_websocket.on_open = on_open
48      fd_websocket.run_forever()
```

Na função *main* as atribuições para execução do programas são definidas, entre elas o cabeçalho da requisição HTTP e a ligação entre a biblioteca e as funções anteriormente definidas. Assim que a função *fd_websocket.run_forever()* é invocada a função *on_open* é executada e envia a primeira mensagem ao servidor, o conteúdo dessa mensagem é uma solicitação de *handshake* com número de identificação de mensagem do cliente(cid) igual a um.

Em seguida o servidor responde com a mensagem - “rid”:1 - informando que a primeira mensagem foi recebida com sucesso. Nesse momento dentro da função *on_message* uma segunda mensagem é enviada com a finalidade de se inscrever para recebimento dos dados de ataque de todo mundo, esta mensagem tem o conteúdo - $'\{“event” : “\#subscribe”, “data” : \{“channel” : “global”\}, “cid” : 2\}'$.

Se tudo correr bem o servidor responde com a mensagem - “rid”:2 - e assim confirmando que o *handshake* foi bem sucedido. Após essa mensagem os dados de ataque começam a ser enviados. A cada mensagem recebida a função *on_message* é chamada e trata o dado recebido o armazenando no banco de dados local.

A cada dois segundos em média uma mensagem de *ping* é enviada pelo servidor para verificar se o cliente da conexão ainda se encontra ativo. Essa mensagem tem o conteúdo - #1. Quando essa mensagem é recebida o programa deve enviar uma mensagem especial de *pong* com o conteúdo - #2, informando assim que ele continua ativo e aguardando por novas mensagens.

Caso a mensagem de *pong* não seja recebida pelo servidor em no máximo dez segundos a conexão é encerrada por ele. Quando isso acontece a função *on_close* é chamada do lado do cliente, finalizando assim a conexão com o servidor e também com o banco de dados local.

O programa foi desenvolvido para executar de forma constante e ininterrupta. Assim sempre que a conexão com o servidor é perdida o programa busca reestabelecê-la o mais rápido possível. Um processo de monitoramento garante que tanto o programa de captura de dados quanto o banco de dados estão sempre executando.

Os dados recebidos por essa fonte de dados são pré-processados pela empresa e limitados. A Tabela 3.1 trás um resumo dos dados que são armazenados no banco de dados acompanhados de exemplos de valores. Através da observação dos dados foi possível obter algumas informações iniciais como a existência de dados redundantes como por exemplo o *dport* e o *svc* que sempre possuem sempre o mesmo valor. O *vector_id* e *type* só tem significado quando utilizados dentro da página da empresa pois são utilizado durante as funções de desenho do mapa.

Mesmo assim todos os dados são armazenados no banco por completude.

Um ponto importante que foi considerado após a implantação do programa no ambiente de execução, foi quantos dados ele poderia processar e armazenar em um período definido de tempo, ou seja qual a vazão de captura da ferramenta. Analisando os dados colhidos durante seis meses é possível afirmar que a ferramenta possuí uma vazão de em média 330 mil dados de ataques capturados e armazenados por dia.

Como não foram coletados dados de outras fontes para comparação, essa vazão foi considerada adequada e representa uma grande massa de dados para processamento.

Tabela 3.1: Descrição dos dados capturados da empresa NorseCorp

Nome Dado	Descrição	Exemplo
_id	Identificador único do objeto no banco de dados	ObjectId("58d1bbbfe2e558522a"),
city	Cidade de origem do ataque	Washington
city2	Cidade onde o alvo do ataque reside	De Kalb Junction
dport	Porta em que o ataque foi executado	25
countrycode	Código do país de origem do ataque	US
countrycode2	Código do país do alvo	US
country	Identificação do país de origem do ataque	US
country2	Identificação do país do alvo	US
latitude	Latitude aproximada da origem do ataque	38.95
longitude	Longitude aproximada da origem do ataque	-77.02
latitude2	Latitude aproximada do alvo	44.48
longitude2	Longitude aproximada do alvo	-75.3
svc	Serviço explorado pelo ataque	25
timestamp	Data e hora no momento da captura do dado	2017-03-21 23:48:15.161973
vector_id	Informação encaminhada pelo servidor para desenho em tela de animação	NumberLong("303241862584")
org	Organização que detêm o bloco de endereços de IP o qual originou o ataque	Microsoft Corporation
type	Informação sobre a ferramenta da empresa que capturou a informação do ataque	ipviking.honey
md5	Número de IP da máquina que enviou o ataque	65.55.169.250

3.1.3 Processamento dos Dados

Posteriormente à captura dos primeiros dados, estudos sobre qual a melhor abordagem para processar esses e extrair alguma informação nova. A busca se concentrou em algoritmos de extração não supervisionada de dados, uma vez que a utilização de algoritmos supervisionados não se apresentava como uma solução viável visto a magnitude do número de dados capturados em um dia.

Assim sendo, após uma pesquisa sobre as vantagens e desvantagens dos principais métodos a escolha recaiu sobre o algoritmo K-Means de clusterização. Com a utilização desse algoritmo era possível identificar grupos baseado na distância euclidiana de forma rápida e simples.

A implementação do algoritmo utilizou a biblioteca de desenvolvimento de aplicações cientificas *SciPy*, descrita na seção 2.3. Dentro dessa biblioteca o algoritmo encontra-se já implementado, bastando agora somente ajustar as entradas ao que é esperado pela biblioteca e projetar os gráficos e resultados.

Alguns ensaios foram realizados, e foi constatado que o tempo para processamento de toda base de dados era inviável. Com mais de 30 milhões de entradas o processamento de todos esses dados leva um tempo proibitivo impedindo que várias consultas sejam realizadas para ajustar e adequar os métodos de processamento.

Dessa forma, antes de começar a aplicar o algoritmo sobre toda a base de dados foi decidido executar o programa somente sobre um grupo de controle com a finalidade de definir quais os melhores arranjos para extração de informação.

O algoritmo K-Means permite calcular a distância entre pontos em um espaço com N dimensões. No inicio a ideia era executar o algoritmo sobre todos os dados, a grandeza de cada dado seria projetada sobre uma dimensão e os pontos distribuídos dentro deste espaço.

Porém, seguindo esta abordagem não seria possível projetar gráficos como resultado para um conjunto maior que 3 dados, tornando possível somente a avaliação dos resultados numéricos. Com o objetivo de explorar primeiro as possibilidades mais simples, um subconjunto de dados foi escolhido para que assim fosse possível o retorno visual da execução do algoritmo.

O problema agora era quais subconjuntos seriam escolhidos, alguns conjuntos iniciais foram definidos baseando-se no conhecimento geral sobre a correlação desses. Um dos primeiros subconjuntos foi o do número de ataques em uma porta distribuído durante as horas de um dia. O segundo foi projetar os ataques por porta distribuídos pela longitude.

Para manter a consistência dos resultados, as amostras de dados foram retiradas de duas formas. Na primeira, os dados eram escolhidos de forma sequencial até um número máximo de registros. A segunda, buscava os registros de forma aleatória dentro da base de dados também respeitando um número máximo de registros pré-definido.

O restante dos subconjuntos foi definido após a execução desses primeiros testes fundamentando os próximos subconjuntos em experimentação orientada pelos resultados obtidos anteriormente.

Com os subconjuntos definidos, o desenvolvimento do programa para o processamento dos dados iniciou-se. O programa desenvolvido para processar os subconjunto está disponível no Apêndice G.

Para apresentação dos resultados do processamento utilizaram-se gráficos e resultados numéricos. Os resultados numéricos foram exibidos por meio da saída padrão do sistema operacional. Essa dupla combinação torna a análise do processamento mais segura e dinâmica, agilizando a construção do conhecimento sobre os resultados.

Os gráficos foram projetados em tela usando a biblioteca *Matplotlib* que compõe a suíte de aplicação do *SciPy*. Essa biblioteca apresenta uma grande gama de opções para visualização das informações, possuindo também suporte a projeções em duas e três dimensões.

Como complemento a essas duas bibliotecas gráficas também foi integrado ao projeto uma extensão da bilioteca *Matplotlib*, a *Matplotlib Basemap Toolkit*, que foi descrita na

subseção 2.3.3, que permite dispor pontos sobre mapas de forma mais simples e com precisão.

3.1.4 Problemas com a Abordagem

Como apresentado anteriormente, alguns problemas foram encontrados durante a implementação da solução proposta. Um dos principais problemas é que esta abordagem é focada em desenvolvimento e armazenamento no mesmo equipamento, assim sua escalabilidade é praticamente inviável. Mesmo a linguagem *Python* possuindo suporte a bibliotecas para processamento paralelo sua implementação não é nativa e possui complexidade elevada já que é preciso fazer de forma explícita a divisão do trabalho entre os processos.

A única forma de acelerar o processamento nesta abordagem é aumentando o número de unidades de processamento e quantidade de memória RAM, tarefa esta que é complexa e com um custo alto.

Outra dificuldade encontrada durante o desenvolvimento do programa, a biblioteca utilizada, *Matplotlib*, exige que os dados estejam em um formato específico para que posam ser processados pela função de clusterização. A conversão dos dados para esse formato tem que ser realizada através de um laço de repetição que atrasa ainda mais a execução do código.

Outra limitação foi a de qual seria a melhor forma de exibir os dados a partir de gráficos. Diversos exemplos foram realizados para definição da melhor configuração dos gráficos em tela. Ajustes nos eixos coordenados e escalas dos gráficos tiveram que ser relizadas para que se pudesse ter pleno entendimento das informações apresentadas.

Outro fator se soma ao tempo de execução do programa de processamento, o tempo para execução do algoritmo K-Means. A biblioteca utilizada implementa o algoritmo de Lloyd [45] para execução, este algoritmo tem a complexidade no tempo de $O(n * K * I * d)$ [45]. Onde:

- n - Quantidade de pontos
- K : Quantidade de *clusters*
- I : Quantidade de interações
- d : Quantidade de atributos

Um conjunto muito grande de dados implica em um conjunto muito grande de pontos. Para um número elevado de pontos a complexidade do algoritmo de Lloyd [45] tende a

ser linear. Entretanto, um conjunto diferente de dados pode levar a diferentes números de interações.

E por fim, um último fator que interfere diretamente no tempo de execução é o tempo de acesso para leitura dos dados. Mesmo o banco de dados *MongoDB* possuindo mecanismos para otimização das operações de leitura, esta operação leva um tempo considerável para acesso dos registros de forma sequencial. O acesso aleatório é ainda mais demorado, visto que não há uma implementação deste banco de dados para busca de registros aleatórios em sua base, cabendo ao programador escrever funções que simulem tal acesso.

Em outra frente há também o custo computacional de processamento e utilização de memória. E é neste ponto que os maiores problemas surgiram. O consumo de memória RAM e de ciclos de processamento do programa é elevadíssimo,

O principal vilão neste caso é banco de dados não relacional *MongoDB*, isso se deve a forma de execução do motor de armazenamento *WiredTiger*. Cinco pontos principais tornam o seu uso difícil para consultas que tem como resposta milhões de documentos. São eles:

- Os dados são comprimidos em disco, porém quando são copiados para memória ficam sempre descomprimidos;
- Por padrão o motor não sincroniza os dados em cada envio, assim os arquivos de *log* também permanecem em memória;
- Ele mantém múltiplas cópias em seu *cache* que fica em memória;
- O próprio banco de dados consome memória para manter conexões, agregações e códigos executando;

Em cada consulta aos dados o banco descomprime milhões de dados e mantêm parte deles em *cache* consumindo toda a memória disponível no computador. Mesmo assim, dada a quantidade de dados armazenados, ele não consegue colocar todos os documentos em memória relizando ainda muitos acessos ao disco reduzindo a eficiência do programa.

Todas essas dificuldades impediram que avanços pudessem ser realizados na extração de informação dos dados. A medida que a quantidade de dados crescia de forma vertiginosa se tornava cada vez mais inviável executar testes sobre o conjunto dos dados para descobrir novas formas de processar tais dados.

Nesta abordagem, cada consulta leva em média cinco minutos para executar sobre uma amostra irrisória das entradas, quando tentativas de executar o processamento sobre todo o conjunto o programa levava mais de trinta minutos para concluir sua execução. A única alternativa seguindo essa linha de abordagem era conseguir outro computador

mais potente para execução das rotinas. Porém, essa solução é impraticável no mundo real visto que a quantidade de ataques continuaria sempre a crescer.

3.2 Segunda Abordagem

Estava claro que uma mudança de paradigma na solução do problema tinha que ser realizada. A impossibilidade de escalar a arquitetura anteriormente descrita ensinou uma importante lição, se a massa de dados a ser processada será grande você deve começar com uma estrutura que possa suportar o processamento e com capacidade de se adaptar ao crescente aumento de dados.

Pensando nisso, foram definidas quatro características gerais que nortearam o desenvolvimento da segunda abordagem, são eles:

- Escalabilidade
- Elasticidade
- Alta disponibilidade
- Adaptabilidade

Seguindo esses conceitos, o sistema atenderá o crescente aumento no número de dados e permitirá diversos tipos de análise e uma extração de dados muito mais ágil, permitindo várias abordagens sobre os dados.

3.2.1 Arquitetura

A arquitetura de *cluster* foi proposta para adequar-se as características definidas, desse modo somente a adaptabilidade não estaria solucionada diretamente pela escolha desta arquitetura. Porém, com a escolha da solução Cloudera para gerenciamento do *cluster* essa última característica também é resolvida, pois este ambiente permite gerenciamento centralizado deste *cluster* com uma grande quantidade de serviços pré-definidos e capacidade de adição de muitos outros de forma simples e com baixo custo.

A escolha pelo Cloudera foi com o objetivo de simplificar a gerência e instalação dos componentes que compõe o *cluster* além de centralizar o controle e monitoramento de todos os serviços e equipamentos presentes. A partir disso, é possível adicionar e remover computadores do *cluster*, e também adicionar e remover serviços em todos os computadores pertencentes. Essas funcionalidades permitem que a execução de programas nesse ambiente sejam elásticos, escaláveis e adaptáveis.

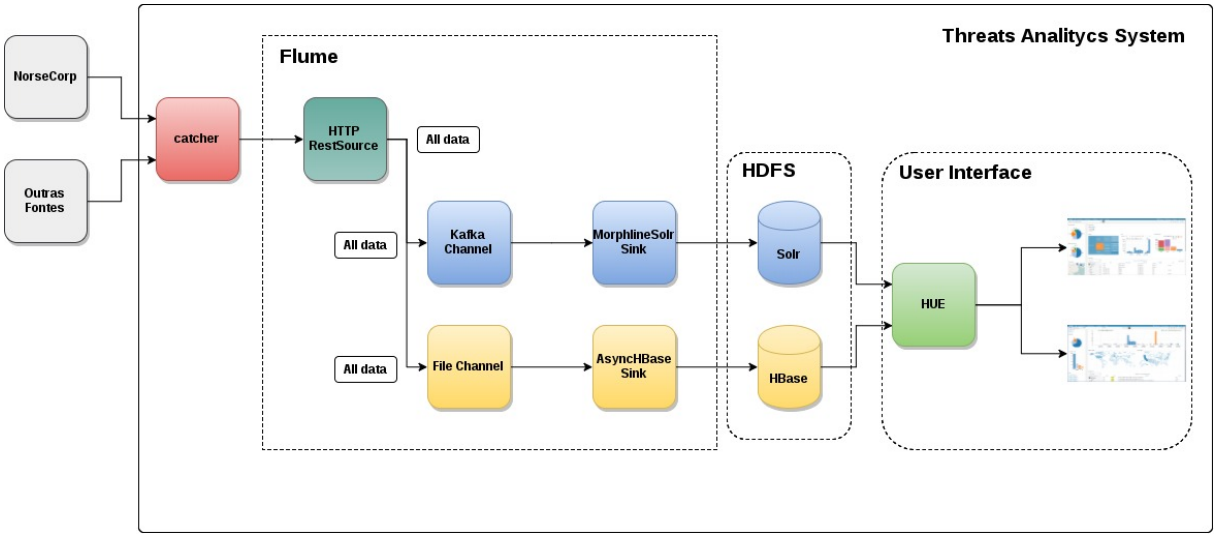


Figura 3.2: Arquitetura proposta para a segunda abordagem.

Para garantir a alta disponibilidade, o *cluster* foi configurado em um serviço de hospedagem da empresa Google que garante funcionamento ininterrupto do sistema.

A Figura 3.2 ilustra a configuração da arquitetura completa para implementação da segunda abordagem. Os dados de ataques são capturados pelo *catcher* que simula uma conexão real com o site que fornece as informação, como por exemplo o site da *NorseCorp*. Esse programa recebe os dados das fontes abertas e cria uma requisição do tipo HTTP *GET* enviando em seguida esta mensagem ao programa *Flume* através de um *source* HTTP.

Este *source* está conectado e replicando os dados a dois *channels*, *Kafka Channel* e *File Channel*. Após os dados serem enviados a estes dois canais, por uma característica do *Flume*, ele só será retirado do *channel* após ter sido entregue em sua respectiva *sink*.

Nesse caso existem duas *sinks*, *MorphlineSolr Sink* e *AsyncHBase Sink*. A primeira processa os dados para inserção desses no programa *Solr*, o segundo os processa para armazenamento do programa *HBase*.

Com todos os dados já inseridos e persistidos, a ferramenta *Hadoop User Experience* (HUE) é responsável por exibir esses dados de forma clara, objetiva e simples. São inúmeras opções de exibição e filtragem dos dados, podendo o usuário em tempo real extrair informações e construir conhecimento prévio sobre uma série de aspectos dos dados de ataques capturados.

3.2.1.1 Captura de Dados (*catcher*)

O programa para captura dos dados foi muito semelhante ao utilizado na primeira abordagem, algumas modificações foram feitas buscando aumentar a eficiência do programa

na captura dos dados. Outra modificação foi a de que em vez de persistir os dados em um banco de dados agora o programa os envia por meio de uma requisição HTTP até o servidor onde o programa *Flume* se encontra em execução.

Além de melhorias no desempenho, uma nova fonte de dados foi inserida na arquitetura para diversificação dos dados, a nova fonte escolhida foi a empresa *LookingGlass Cyber*. Assim como a *NorseCorp* essa empresa disponibiliza parte dos ataques infringidos a seus clientes em um site, dessa forma é possível interceptar tais dados e adiciona-los as bases internas da arquitetura.

Os dados capturados nessa nova fonte de dados são expostos na Tabela 3.2 bem como um exemplo de cada um deles.

Tabela 3.2: Descrição dos Dados LookingGlass Cyber

Nome Dado	Descrição	Exemplo
asn	Sistema Autonomo (AS) de onde partiu o ataque	AS23693 PT. Telekomunikasi Selular
botnet	Nome da *botnet* que infectou o computador	Sality
city	Cidade de onde partiu o ataque	New York
countrycode	Código do país de origem do ataque	US
organization	Empresa responsável pela AS de onde partiu o ataque	Telkomsel
variant	Variação da *botnet* principal	
country	País onde se encontra o computador atacante	US
latitude	Latitude aproximada da origem do ataque	38.95
longitude	Longitude aproximada da origem do ataque	-77.02
time	Data e hora do ataque	2017/12/12T08:30:12Z

Para melhor coesão, para cada empresa foi gerado um programa em *Python* para realizar a captura dos dados. Esses programas, como já mencionado, foram baseados no programa de captura da primeira abordagem, porém modificados para melhoria da captura e armazenamento dos dados.

O programa utilizado para captura dos dados da *NorseCorp* pode ser encontrado no Apêndice E. Enquanto o programa para captura dos dados da *LookingGlass Cyber* se encontra no Apêndice F.

As principais modificações para o programa de captura foram com o objetivo de melhorar a vazão na captura dos dados. Durante a medição da quantidade de ataques capturados na primeira abordagem uma ponderação foi feita sobre se aquela realmente

era a quantidade de dados enviada pelo site ou se algum gargalo durante o processamento desses estaria limitando a quantidade capturada.

Após as modificações foi possível constatar um aumento de em média 10% no número de ataques diários capturados a partir da página da *NorseCorp* saltando estes de 330.000 para por volta de 363.000 ataques diários. Esse aumento foi considerado significativo já que esse número representa uma massa substancial de dados.

A vazão de dados capturados da *LookingGlass Cyber* é bem superior ficando por volta de um milhão de ataques diários capturados. O motivo disso é que a empresa envia muito mais dados através de seu site. Como essa segunda fonte de dados foi inserida somente nesta segunda abordagem não há como realizar um comparação de desempenho.

3.2.1.2 Configuração do Flume

O arquivo de configuração do Flume é apresentado no Apêndice A e representa exatamente a arquitetura já descrita anteriormente e ilustrada na Figura Figura 3.2. O arquivo segue as especificações da ferramenta *Flume*, vale salientar que foram mantidas todas as configurações padrão durante a instalação da ferramenta utilizando o Cloudera.

Outros arquivos também são necessários para complementar a configuração do *Flume*, como por exemplo o arquivo que configura o *framework Morphlines* que é usado para transformar os dados durante a ingestão. Ele é conectado no *Flume* como uma *sink* que recebe os dados os trata e envia até o *SOLR*. Esse arquivo de configuração é disponibilizado no Apêndice B.

Por último, temos o arquivo que configura a *sink* que persiste os dados no sistema HBase, para configurar esta haviam duas opções o *SyncHBASESink* ou o *ASyncHBASE-Sink*. O primeiro atua de forma síncrona, ou seja ele só recebe o próximo dados depois de receber a resposta positiva, essa característica poderia gerar um gargalo durante o armazenamento dos dados o que poderia ocasionar uma diminuição da taxa de captura ou até mesmo um problema de memória.

A alternativa escolhida foi, utilizando a segunda opção que grava os dados no sistema de forma assíncrona recebendo novos dados mesmo que ainda não tenha confirmado o armazenamento do anterior. Esse arquivo é um programa em *Java* que roda toda vez que um novo dado é recebido pela *sink*. ele insere o nome da coluna e o separador específico. Assim estrutura os dados de forma que eles sejam exibidos em suas colunas corretamente. Caso não fosse realizado este passo todos os dados permaneceriam em uma mesma coluna dificultando a filtragem dos dados. Este arquivo esta disponível no Apêndice D.

3.2.1.3 Configuração *SOLR*

Para configurar o sistema de indexação *SOLR* é necessário definir um esquema utilizando um arquivo de configuração e executar comando de forma que a coleção a que os dados vão pertencer seja criada. O arquivo de configuração é descrito seguindo a linguagem *XML* e tem por objetivo definir os campos e seus tipos, assim a ferramenta pode indexar tais dados de forma correta permitindo buscas e extração de informação de forma mais rápida.

O arquivo de configuração possui todos os campos que são recebidos pelo Flume, seus tipos foram escolhidos de forma a facilitar o processamento dos dados, o arquivo é apresentado no Apêndice C. Cada uma das fontes de dados possui seu próprio arquivo de *schema*, isso é necessário para que o SOLR possa indexar as duas bases em ambientes diferentes. Dessa forma a análise dos dados é feita de forma dissociada.

Após o arquivo ser posicionado em um diretório qualquer do sistema os seguintes comandos devem ser executados para se configurar a nova coleção.

```
solrctl instancedir --generate caminho_para_arquivo/nome_da_coleção

solrctl --zk ip_zookeeper:2181/solr instancedir
--create nome_da_coleção caminho_para_arquivo/nome_da_coleção

solrctl --zk ip_zookeeper:2181/solr collection --create nome_da_coleção -s 1
```

O primeiro comando gera um diretório padrão com todos os arquivos necessários para que se crie uma coleção de índices. Os arquivos são criados com valores padrão. O segundo comando realiza o carregamento das informações, contidas no diretório que foi criado, para o SolrCloud tornando disponível aquela coleção. Por fim, o último comando instância a coleção de índices tornando essa pronta para receber e indexar os dados recebidos.

3.2.1.4 Configuração HBase

A configuração do sistema HBase é uma das mais simples bastando somente realizar a criação de uma nova tabela para o armazenamento dos dados. Isso pode ser feito executando os comandos:

```
hbase shell -n
create <nome da tabela>,<collumm family>
```

O nome em *collumm family* descreve o prefixo das colunas que é comum a todas as tabelas com o mesmo *collumm family*.

3.2.2 Processamento dos Dados

O processamento dos dados sempre é realizado durante a ingestão deles no sistema. Esse método foi escolhido para que o processamento dos dados ocorresse de forma rápida, permitindo que a posterior obtenção das informações resultantes acontecesse quase que em tempo real.

Porém, é claro que outras técnicas de processamento poderiam ser aplicadas ao sistema, por esse motivo os dados também são persistidos no sistema HBase. Dessa forma em um momento futuro é possível recuperar esses dados e transformá-los para assim gerar novas informações. Entretanto, tal processamento seria realizado de forma desvinculada do primeiro processamento que continuaria sempre a executar.

A indexação durante o armazenamento dos dados é o que torna a análise do dados ágil sendo essa característica uma das mais importantes do sistema. Dessa forma é possível utilizar um sistema de visualização de tabelas e a transformação dos dados em gráficos, sem que seja necessário um grande tempo aguardando pelas informações.

O sistema *SOLR* executa essa tarefa com maestria indexando e armazenando os dados em sua própria base de dados que se encontra por sua vez sobre um sistema de arquivos distribuídos *Hadoop Distributed File System* (HDFS), garantindo assim a integridade e disponibilidade dos dados.

O processamento sobre os dados já armazenados não foi desenvolvido por uma questão de tempo. Para a realização dele foi escolhido o sistema Spark para execução das rotinas de MapReduce entre outras.

As vantagens do Spark são muitas, entre elas se pode citar a facilidade de aplicação de métodos estatísticos sobre os dados, facilidade de aplicação de métodos de aprendizado de máquina, velocidade no processamento dos dados e integração simples com o HBase e SOLR.

3.2.2.1 Visualização

Toda essa estrutura por mais robusta que seja, não representaria um grande avanço se não fosse possível examinar as entradas em busca de informação.

Para interação entre o usuário e os dados o sistema HUE foi escolhido pois sua interface é amigável e robusta. Com esse é possível exibir os ataques utilizando vários recursos gráficos como tabelas, mapas, gráficos de barra e pizza.

Além disso, ele também permite que sejam feitas filtragens e seleções sobre os dados, simplificando e muito a extração de informações. A sua principal vantagem reside nisso, a abstração que ele gera sobre todo o resto da arquitetura desenvolvida. Sendo possível

que uma pessoa com pouco conhecimento em ferramentas e métodos de *Big Data* possa olhar, entender e gerar conhecimento interagindo com a aplicação.

3.2.3 Considerações Finais

No inicio do desenvolvimento da solução, a hipótese que se tinha era que a massa de dados capturadas poderia ser analisada utilizando somente um computador pessoal sem grandes problemas. Esse engano impediu a evolução do desenvolvimento dos programa e análise dos dados, todavia ele foi identificado cedo o bastante para ser possível sua correção. Sendo usado como prova de conceito e validador para a captura dos dados. O desenvolvimento de uma arquitetura robusta e com possibilidade de expansão se mostrou ser primordial para o tratamento de tal volume de dados, os ganhos de sua implementação permitiram sair de um sistema simples que levava bastante tempo para fornecer informações até um sistema que possibilita a extração de informações em tempo real.

Num ambiente tão dinâmico e vasto como a Internet essas características são indispensáveis para permitir extrair informações com efetividade.

Capítulo 4

Resultados e Análise

4.1 Resultados

Antes de começar a expor e analisar os resultados faz-se fundamental uma importante observação sobre os dados capturados, assume-se que esses sejam completamente verdadeiros. Mesmo sendo necessária uma análise estatística rigorosa para confirmação da veracidade desses o objetivo deste trabalho não foi focado em executar tais análises mas sim definir aspectos metodológicos de coleta e análise de dados de ataques a partir de fontes abertas.

Cabe ressaltar, mesmo que os dados capturados tenham sofrido algum tipo de tratamento estatístico ou que esses não reflitam distribuições reais, neste trabalho esses fatores não foram considerados assim assume-se que todos os eventos capturados de fato ocorreram em algum momento.

4.1.1 Primeira Abordagem

Utilizando a arquitetura da primeira abordagem, foi possível obter uma série de resultados expressivos. Esses resultados são descritos nesta seção.

Uma das primeiras escolhas para construção dos gráficos foi projetar o número das portas atacadas por um determinado período de tempo. Os dados foram escolhidos de forma aleatória na base buscando que assim se representasse o comportamento geral dos dados.

A Figura 4.1 exibe os resultados obtidos, nessa figura cada ponto representa um ataque realizado. As cores simbolizam a que grupo pertence aquele ataque, nesse gráfico o montante de dados só foi dividido em dois grupos. Os quadrados verdes indicam os centroides dos conjuntos. No eixo X essas horas do dia estão sinalizadas, e no eixo Y o número de porta do ataque.

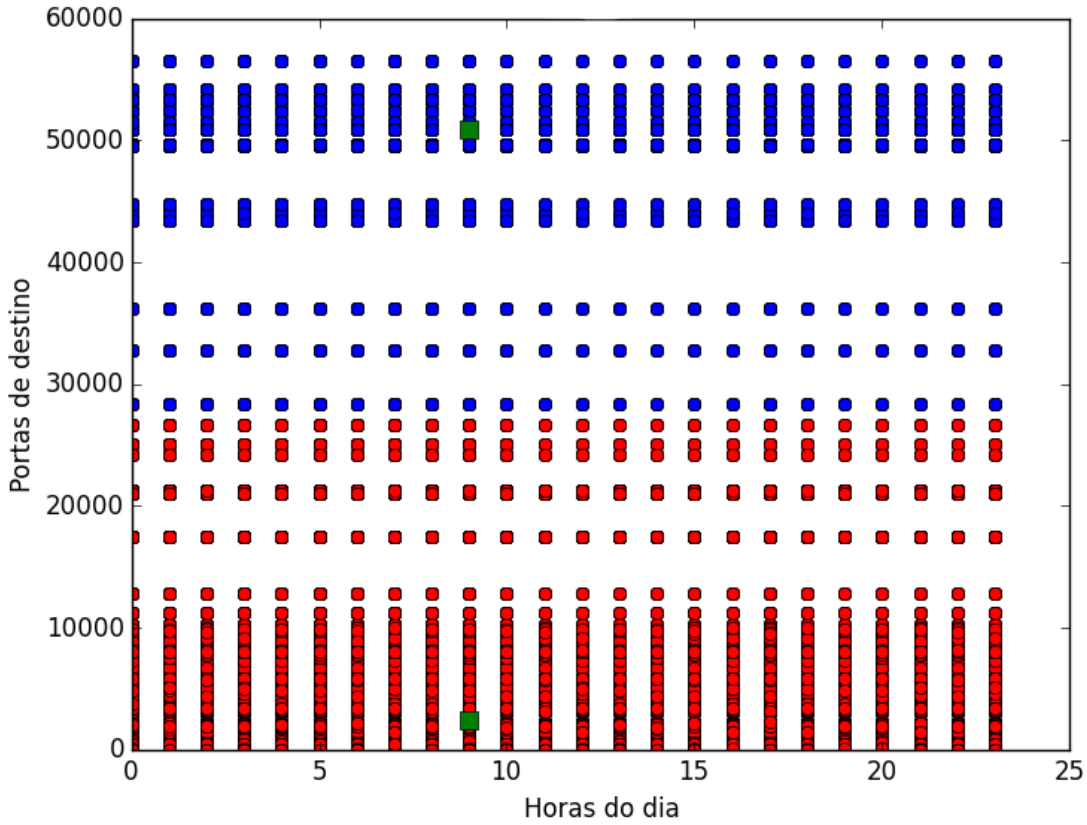


Figura 4.1: Número da porta relacionado a hora do dia.

Verificando a Figura 4.1 é possível perceber que a distribuição dos ataques em cada porta é constante durante o dia. Para confirmação das informações também foram escolhidos dados em intervalos específicos, porém a distribuição dos pontos permaneceu inalterada.

Essa informação, a primeira vista, pode ser considerada desanimadora, como de fato foi. Perceber que um padrão simples se mantêm durante todas as horas do dia não parece uma grande descoberta e nem causa um impacto tão grande, porém esta observação de fato é importantíssima pois corrobora a teoria de que os ataques realizados de forma massiva são massivamente realizados por ferramentas automatizadas.

Como seria improvável que um ser humano conseguisse manter a distribuição de ataques constante durante um longo período de tempo, as informações sobre ataques constantes na Figura 4.1 sinalizam que várias ferramentas automáticas mantêm uma taxa constante de execução, o que por consequência gera a projeção observada.

Esse fato permite que se observe a frequência dos ataques para extrair ainda mais informações. Calculando essa pode-se agrupar os ataques a fim de se definir a qual *botnet*

eles pertencem. Como cada programa tem peculiaridades pertencentes somente a ele essa opção parece viável quando agregada a outros fatores, além de que após confirmar que aquele dispositivo dispara ataques de forma automática é possível classificá-lo como um atacante e limitar suas ações no ativos da organização.

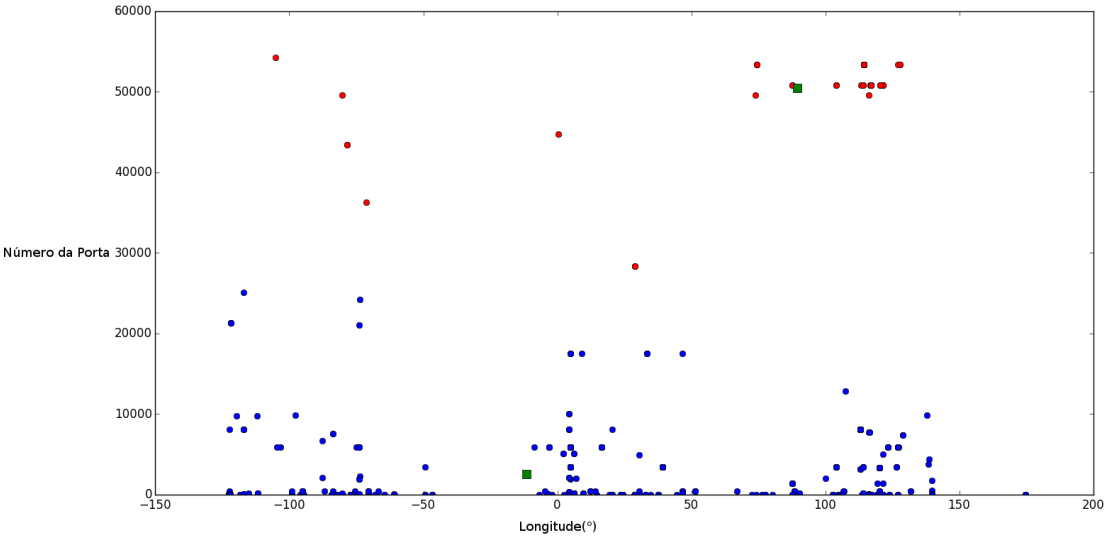


Figura 4.2: Longitude do atacante relacionado a porta escolhida para o ataque.

O próximo resultado foi obtido utilizando uma amostra de 100.000 dados escolhidos aleatoriamente na base. As informações escolhidas para composição dos eixos X e Y do gráfico foram, longitude do atacante e porta atacada. O resultado pode ser visto na Figura 4.2. Nela é possível perceber que ao executar o algoritmo *K-Means* dois grupos são facilmente identificados e sinalizados pelas cores azul e vermelha, um de portas com valores altos e outro grupo com portas de valores baixos. O centro de cada um dos grupos é representado também por um quadrado verde.

Esse resultado de certa forma foi um dos mais expressivos obtidos durante a primeira abordagem, ele demonstrou que era possível a partir de uma base de dados limitada e extremamente simples enriquecer esses de forma a obter informações novas. Observando o gráfico representado na Figura 4.2 evento chama atenção, o centroide do conjunto de portas com valores altos se encontra deslocado para a direita, entre a longitude de 75° e 150°.

Enquanto a distribuição de ataques em portas menores que 10.000 permanecia mais ou menos constante, as portas com valores acima de 40.000 se concentram nas longitudes mais ao leste.

No mapa do mundo representado na Figura 4.3 foi desenhada uma elipse entre as longitudes observadas anteriormente. Vendo os países que se encontram dentro desta é possível encontrar vários países conhecidos por seu papel em ataques em rede, tais como China, Rússia, Paquistão e Índia.

Assim sendo, os resultados se mostraram no caminho certo. Agora bastava saber exatamente a partir de quais países esses ataques singulares se originavam. Como a informação contida na Figura 4.2 não trazia informações sobre a latitude dos ataques um novo gráfico deveria ser projetado.

Figura 4.3: Mapa do mundo com Latitudes e Longitude com detalhe (Fonte: [75]).

Com o objetivo de corrigir essa limitação, que o gráfico representado na Figura 4.4 foi gerado, nele se buscava projetar tanto a latitude como a longitude dos ataques além é claro de conservar a distribuição por frequência dos dados. Esta combinação de fatores conseguiu ser alcançada e é apresentada nele onde a quantidade de ataques em uma determinada localização é proporcional ao raio dos círculos.

As cores dos círculos novamente significam a qual grupo eles pertencem, os círculos verdes são os centroides dos conjuntos. Para o cálculo da frequência dos ataques as porta com valores acima de 40.000 foram agrupadas, dessa forma não é possível perceber na Figura 4.4 as frequências individuas das portas.

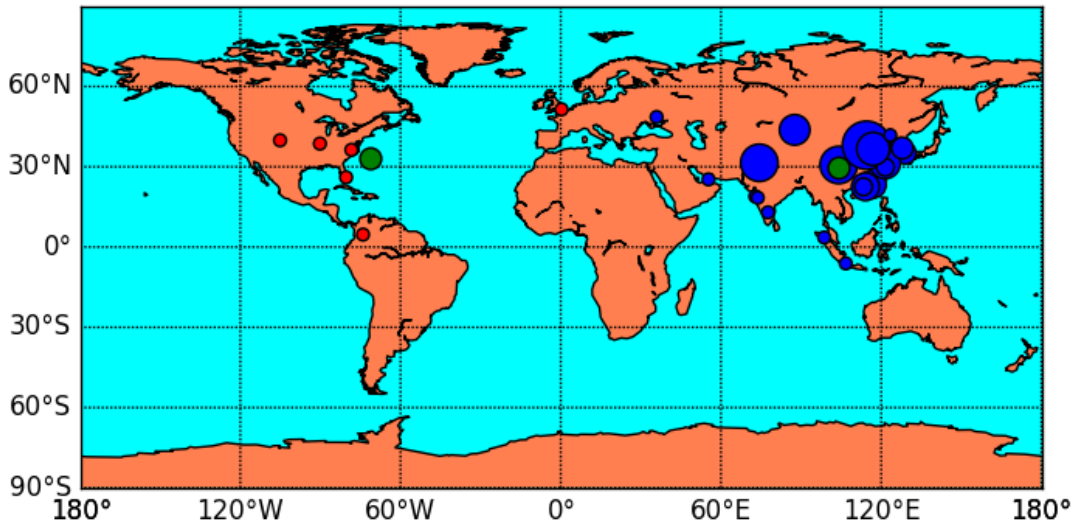


Figura 4.4: Frequência dos ataques dispostos em um mapa.

Agora ficava evidente a origem dos ataques, bem como quais países se utilizam dessa porta com maior frequência, sendo eles China, Paquistão, e Cazaquistão. Com essa nova informação em mãos bastava agora corroborar essa informação obtida através dos dados com outras fontes.

Realizando uma busca rápida, vários fóruns e sites surgiram relatando o interesse de atacantes chineses em portas com valores acima de 40.000 com destaque para as portas 50.866, 50.864 e 54.413 [76, 77] que foram as portas encontradas na consulta exibida na Figura 4.2. Outra informação que surgiu durante as pesquisas foi um possível *backdoor*[1] em roteadores *Netis* de fabricação chinesa a partir da empresa *Netcore Group* [79], através da porta 54.413 [77] os atacantes poderiam tomar total controle do dispositivo sem nenhum conhecimento de seu proprietário.

Essas referências corroboram o resultado obtido através da observação dos dados capturados, e além disso trazem novas informações sobre quais países estão utilizando dessas vulnerabilidades. Isso é importante pois permite auditoria e rastreabilidade para confirmar a origem dos ataques, sabendo assim quais são os países que representam mais riscos para as organizações.

Durante a extração de informações da base ocorre uma série de ataques de *Distributed*

[1]Backdoor é um programa que permite o retorno de um invasor a um computador comprometido, por meio da inclusão de serviços criados ou modificados para este fim.
Pode ser incluído pela ação de outros códigos maliciosos, que tenham previamente infectado o computador, ou por atacantes, que exploram vulnerabilidades existentes nos programas instalados no computador para invadi-lo. [78]

Denial of Service (DDoS)[2]. em nível global, tais como o *Mirai* e o *ransomware*[3]. *WannaCry*. Logo uma nova alternativa foi percebida, agora era possível plotar gráficos para analisar os acontecimentos durante esses períodos.

Para construção dos gráficos havia um problema, a quantidade de ataques capturados em um dia sofria variações. Isso se deve a variações no envio dos dados pela empresa ao programa de captura, e também a intermitências no funcionamento do programa. Como a infraestrutura da primeira abordagem não dispunha de alta disponibilidade o programa com certa frequência perdia a conexão com a rede ou até mesmo encerrava por falta de energia.

Como essa variação do número de ataques não estava relacionada ao comportamento normal dos dados ela deveria ser contornada. A solução escolhida foi utilizar valores proporcionais dos ataques, ou seja dividir o número de ataques com características desejadas e dividir esse valor pelo total de ataques no dia. Dessa forma esperava-se que essas inconsistências não prejudicassem a análise dos dados.

O resultado pode ser visto nas Figuras 4.5 a 4.6. Vale salientar que esses gráficos foram construídos utilizando os ataques em portas utilizadas pelo *Mirai* para invadir e lançar ataques, são elas porta 7547, 5555, 37777, 6789, 81 [80]. Essas portas por sua vez não são alvo constantes de ataques na internet. Assim o objetivo desenhando esses gráficos era ver se era possível observar algum aumento no número de ataques nos dias em que o ataque foi percebido gerando grande volume de informações.

A Figura 4.5 mostra a porcentagem de ataques com características do *Mirai* distribuídos ao longo de quase três meses. Um fato interessante de ser observado é de que esses ataques nunca superam $0,25\%$ dos ataques totais, um número pouco significativo representando cerca de 750 ataques por dia.

Outra informação chama atenção, no primeiro dia(21/03) apresentado no gráfico da Figura 4.5 o percentual é bem menor do que nos demais dias. Isso levou a duas proposições, a frequência dos ataques era menor antes daquele dia e começou a aumentar, ou somente neste dia houve uma variação?

Para responder a essa dúvida, outro gráfico foi construído, dessa vez utilizando também dias anteriores ao dia que apresentou variação(21/03). Esse gráfico está na Figura 4.6.

[2]Negação de serviço, ou DoS (Denial of Service), é uma técnica pela qual um atacante utiliza um computador para tirar de operação um serviço, um computador ou uma rede conectada à Internet. Quando utilizada de forma coordenada e distribuída, ou seja, quando um conjunto de computadores é utilizado no ataque, recebe o nome de negação de serviço distribuído, ou DDoS (Distributed Denial of Service)[78]

[3]Ransomware é um tipo de código malicioso que torna inacessíveis os dados armazenados em um equipamento, geralmente usando criptografia, e que exige pagamento de resgate (ransom) para restabelecer o acesso ao usuário. O pagamento do resgate geralmente é feito via bitcoins [78]

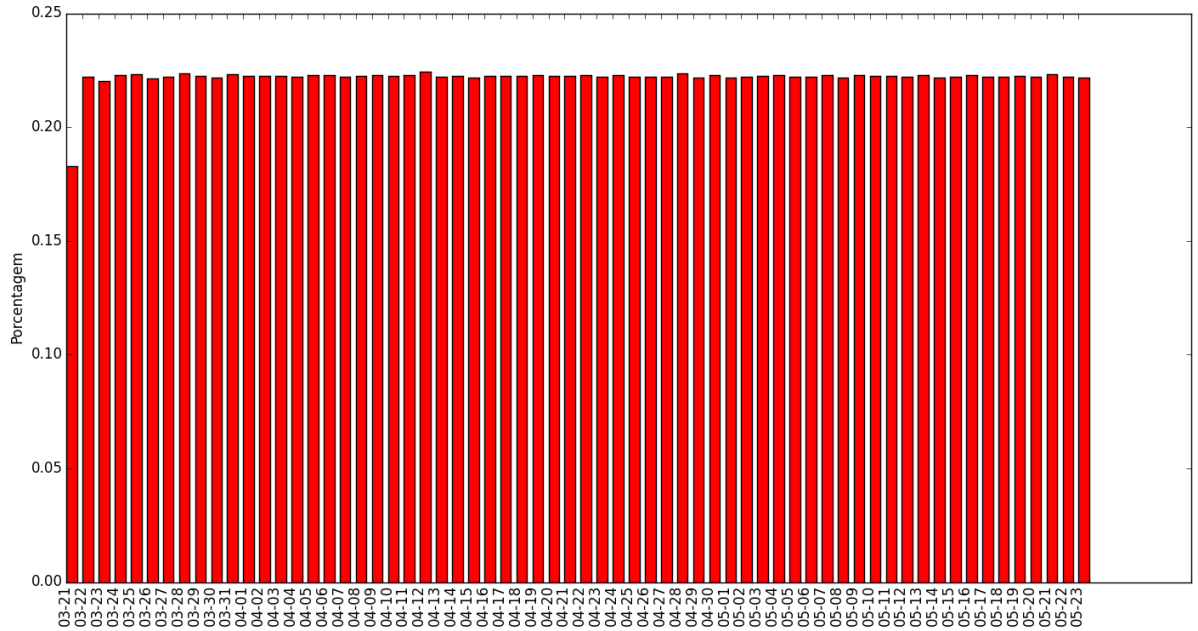


Figura 4.5: Porcentagem de prováveis ataques da *botnet Mirai* por dia.

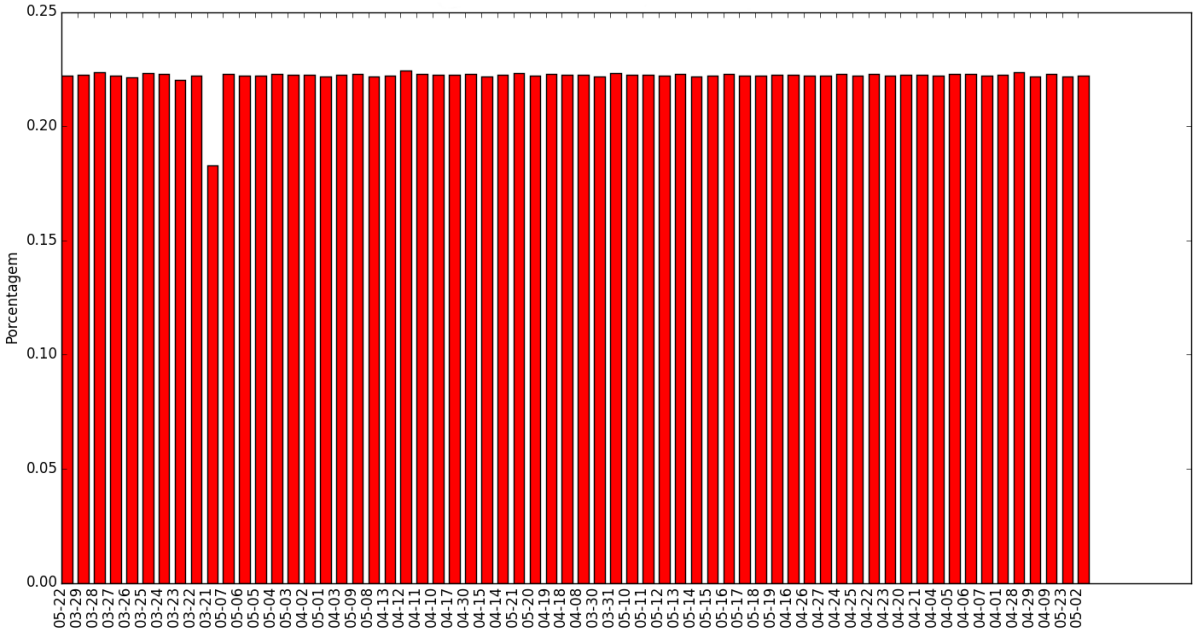


Figura 4.6: Porcentagem de prováveis ataques da botnet *Mirai* por dia, período estendido.

A partir da Figura 4.6 fica claro que a quantidade de ataques anteriores ao dia 21/03 apresentavam também porcentagens muito próximas as dos dias subsequentes ao dia 21/03. Logo a proposição de que a frequência dos ataques era menor antes daquele dia e começou a aumentar só depois foi descartada.

Os dois resultados ilustrados nas Figuras 4.5 a 4.6 foram suspeitos. O primeiro ataque do *Mirai* alcançou um pico de 623 Gbps de tráfego, um dois maiores já registrados por ataques de DDoS. Dessa forma, supondo que as vulnerabilidades exploradas por ele não eram utilizadas anteriormente, era esperado uma grande variação no número de ataques nestas portas.

Um dos fatores que pode ter levado a uma constância nessa distribuição, diz respeito aos clientes da *NorseCorp*. Como a empresa só pode acessar e enviar informações de seus clientes é possível que eles não tenham enfrentando muitos ataques do *Mirai*. Outro fator menos provável seria o de que a empresa não externa dados reais em sua ferramenta, o que seria muito prejudicial para a imagem da companhia caso essa assertiva fosse verdadeira.

Outro questionamento também foi levantado depois de se analisar o gráfico da Figura 4.6, para determinar a porcentagem de ataques prováveis do *Mirai* foi feita uma união entre os ataques ocorridos em todas as portas sinalizadas como sendo de uso da ferramenta. Mas será que a quantidade de ataques mudaria se cada porta fosse observada em separado?

Antes de responder esta pergunta, um estudo sobre as reservas de cada uma das portas foi feita, o objetivo era compreender melhor em quais vulnerabilidades o *Mirai* se baseava. É possível na Tabela 4.1 perceber que as principais portas utilizadas pela *botnet Mirai* não pertencem a nenhum protocolo muito conhecido. O protocolo *DSL Forum CWMP* é utilizado por operadoras de rede que buscam configurar remotamente o roteador de seus clientes, já o *Generic Security Standard Application Programming Interface* (GSS-API) é um protocolo da *Oracle* para criptografia de transmissão entre dois nós em uma rede.

O protocolo registrado na IANA para a porta 5555 não tem uma definição, sendo utilizado de forma informal por diversos outros protocolos. Sendo assim os dois outros protocolos que merecem uma atenção mais apurada, assim foi feita uma busca rápida em busca de vulnerabilidades que poderiam estar comprometendo essas portas. Não foi difícil encontrar essas vulnerabilidades, como descrito pelo site `arstechnica.com` [81] os roteadores fornecidos pelas empresas *Deutsche Telekom* e *Eircom*, localizadas na Alemanha e Irlanda respectivamente, possuem uma vulnerabilidade no protocolo *DSL Forum CWMP ou TR-069* o que permite que invasores tomem conta deste dispositivos.

Para a porta 6789 não foram encontrados resultados expressivos que explicassem sua utilização pela *botnet*.

Em posse das informações sobre as portas utilizadas, pode-se agora procurar maneiras de responder a questão levantada anteriormente, a quantidade de ataques mudaria se cada porta fosse observada em separado?

Para responder essa pergunta, o gráfico representado na Figura 4.7 foi plotado. Nele é possível ter uma visão em três dimensões, no eixo Z as porcentagens de ataques em cada

Tabela 4.1: Reserva definida pela *Internet Assigned Numbers Authority* (IANA) para as portas descritas

Porta	Protocolo	Reponsável
7547	DSL Forum CWMP	Anton Okmianski
5555	Personal Agent	Jackie Wu
37777	Nenhum	Nenhum
6789	GSS-API	Oracle
81	Nenhum	Nenhum

porta agora em separado, no eixo X esta representado o número da porta e finalmente no eixo Y os dias analisados.

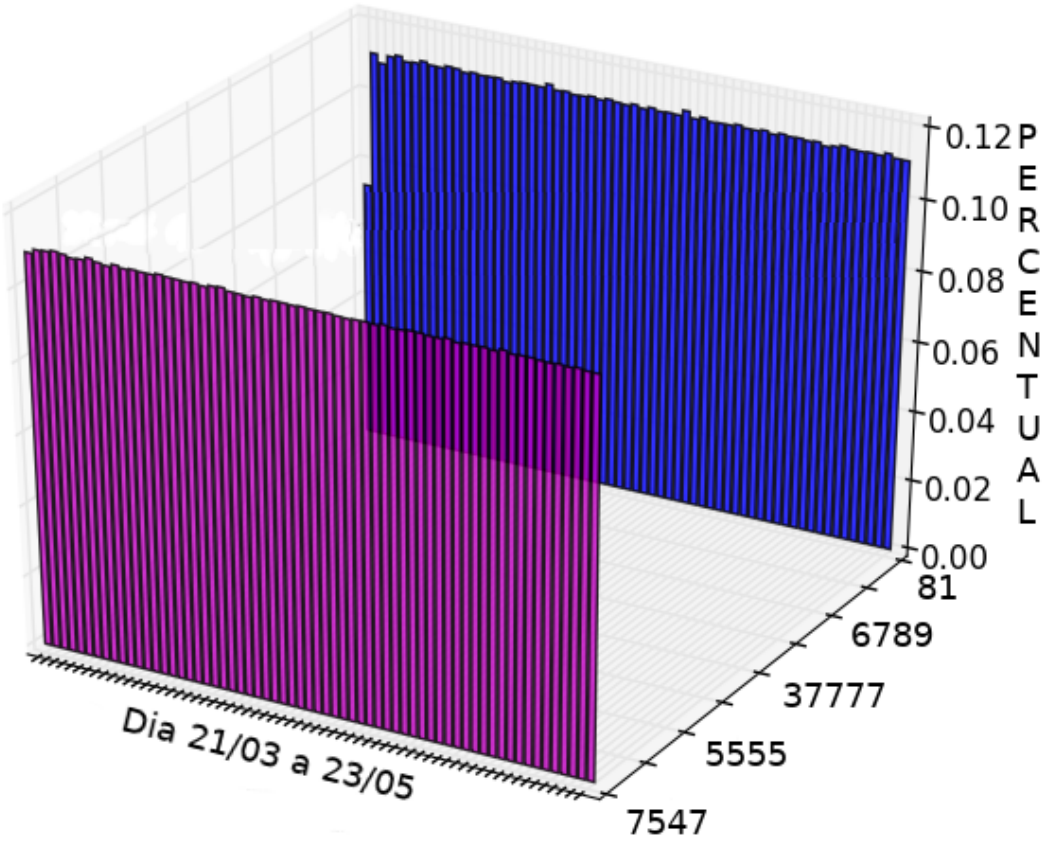


Figura 4.7: Porcentagem de prováveis ataques da *botnet Mirai* por dia separados por porta.

Com esta visão fica evidente que, somente em duas portas há ataques que podem ser quantificados, a porta 7547 e 81. Também é possível visualizar que a distorção na quantidade de ataques do dia 21/03 somente ocorre na porta 81 permanecendo a porta 7547 com sua distribuição de ataques durante os dias analisados inalterada.

Assim, pode-se concluir que, para essa amostra de dados somente essas duas portas foram alvos. Novamente o comportamento dessa distribuição pode ser explicado em função específica dos clientes da companhia.

Os gráficos para análise do *ransomware* *WannaCry* não foram muito diferentes, como pode ser visto na Figura 4.8. Porém, nesse caso há um outro ponto a se considerar já que este neste tipo de ataque, particularmente falando do *WannaCry*, as portas utilizadas foram 137, 138, 139 e 445. Sendo que, essas portas são usadas principalmente pelo sistema operacional *Windows* em seu protocolo de compartilhamento de arquivos o *Server Message Block* (SMB).

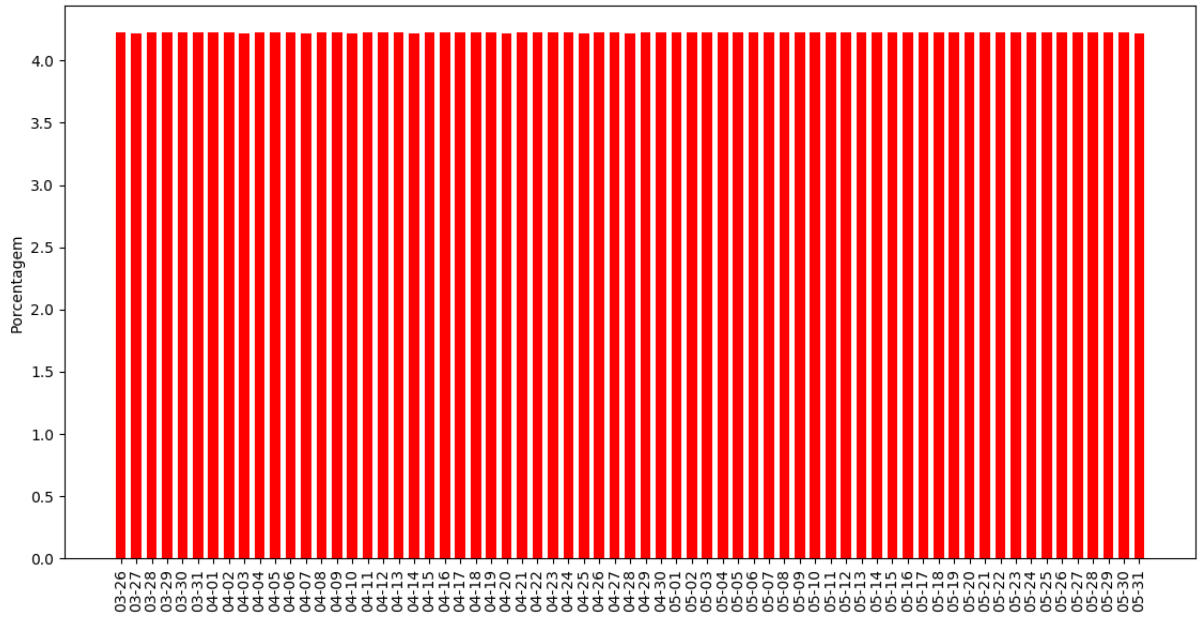


Figura 4.8: Porcentagem de prováveis ataques do *WannaCry* por dia.

Como as medidas de segurança tomadas por esse sistema não são completamente adequadas muitas ferramentas maliciosas fazem uso desta porta. Dessa forma, os dados dos ataques nas portas pesquisadas não são uma boa fonte para quantificar o impacto da ferramenta. Não sendo encontrada outra forma de realizar a busca por outras características do *WannaCry* para construção de outros gráficos.

Não é difícil encontrar descrições de vulnerabilidades que exploram especificamente essa porta. Segundo a *Akamai* em 2014 o volume de ataques na porta 445 representava 15% de todos os ataques registrados pela empresa [27]. Ficando assim evidente que antes mesmo do *ransomware* o volumes de ataque nessa porta era bastante elevado.

4.1.2 Segunda Abordagem

4.1.2.1 Apresentação do Painel de Ferramentas

Se durante a primeira abordagem poucos resultado puderam ser demonstrados por conta do tempo levado ao processamento destes, nessa abordagem o panorama é completamente diferente. Uma infinidade de possibilidades de visualização é disponibilizada, agora é possível aplicar uma série de filtros e tratar os dados em tempo muito superior.

Demonstrar todos os resultados que poderiam ser obtidos com a esta abordagem levaria a um imenso documento, sendo assim somente alguns destes resultados serão exibidos aqui.

Assim que é feito o acesso ao sistema o painel de ferramentas é exibida, na Figura 4.9, essa página é a primeira página quando se seleciona a *dashboard* correspondente a nossa base de indexação na plataforma HUE. Nela já é possível em uma primeira observação visualizar quais portas são mais atacadas no gráfico circular a esquerda nomeado como *target_port* e também a partir de quais países se originam o maior número de ataques olhando para o mapa nomeado como *Marker Map*.

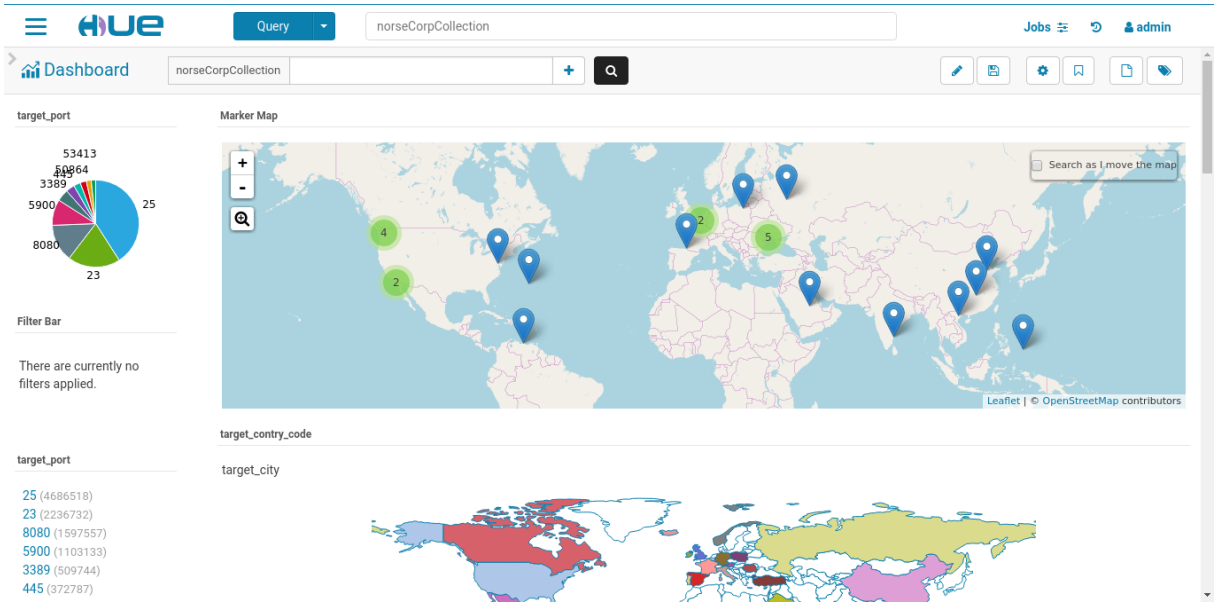


Figura 4.9: Painel de Ferramentas de Análise exibido pela ferramenta HUE.

Abaixo do gráfico circular é possível observar quais filtros estão ativos para aquela visualização olhando no campo *Filter Bar* no caso da Figura 4.9 não há nenhum filtro aplicado. Rolando a página para baixo é possível continuar a observar as opções de indexação, na Figura 4.10 pode-se ver opções de filtragem de portas a esquerda nomeados também como *target_port*. Com essa opção é possível mostrar na página somente os ataques que tem como alvo as portas selecionadas.

Abaixo dessa opção também há a possibilidade de se filtrar os países a partir do qual partem mais ataques. Novamente os dados mostrados sejam somente aqueles que tem aqueles países selecionados como origem dos ataques.

À direita tem-se os países e cidades que recebem o maior volumes de ataques. Os países são identificados pelas cores e logo abaixo está a lista das cidades que mais recebem ataques para estes países, uma para cada país. A correspondência entre a cidade e o país é dado pela cor no quadrado a esquerda de cada cidade.

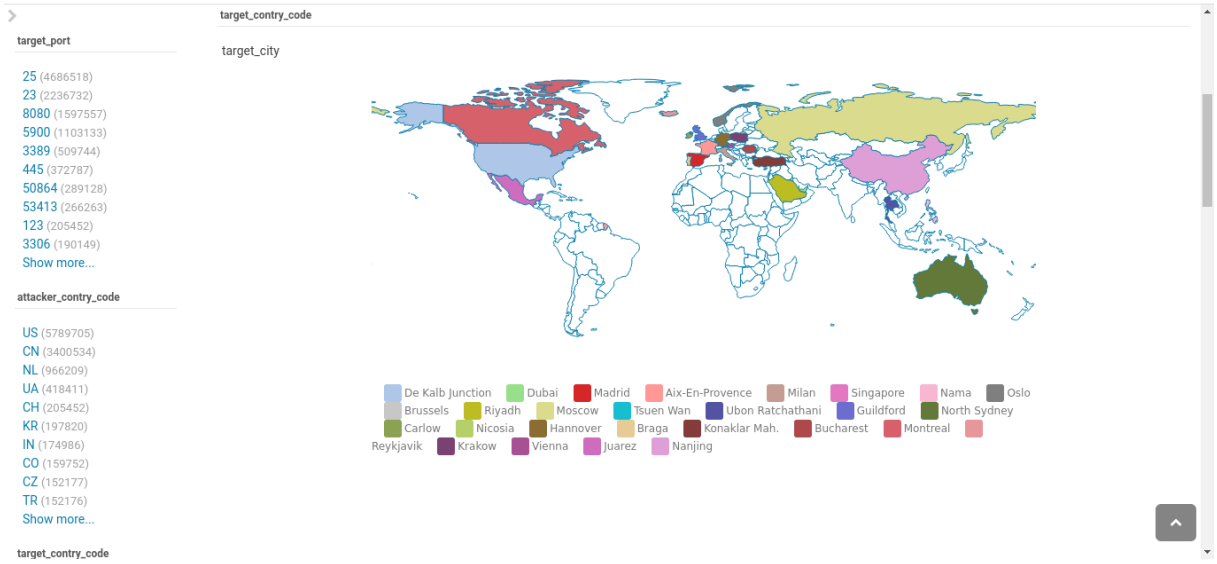


Figura 4.10: Painel de Ferramentas de Análise com detalhe para exibição de países alvo.

Seguindo pela tela inicial, há mais dois campos que podem ser vistos na Figura 4.11, um para realizar a filtragem pelo país alvo e outro para visualização das organizações que mais hospedam máquina das quais partem os ataques. Selecionando o filtro de país de origem denominado na tela por *target_country_code* pode-se exibir somente os países alvos selecionados.

No gráfico de barras são mostrados as dez maiores organizações responsáveis pelas máquinas que dispararam os ataques, o número de ataques é mostrado a esquerda. Devido a resolução da tela e a extensão dos nomes das organizações estes não estão perfeitamente alinhados, porém é possível acompanhar a lista de nomes em ordem com a ordem das barras.

As duas últimas opções de para análise dos dados estão ilustradas nas Figuras 4.12 a 4.13. A Figura 4.12 representa os países onde se encontram o maior número de atacantes e é denominada *attacker_country_code*, logo abaixo se encontra a cidade naquele país que concentra o maior número de ataques. A relação entre cidade e país é definida pelo

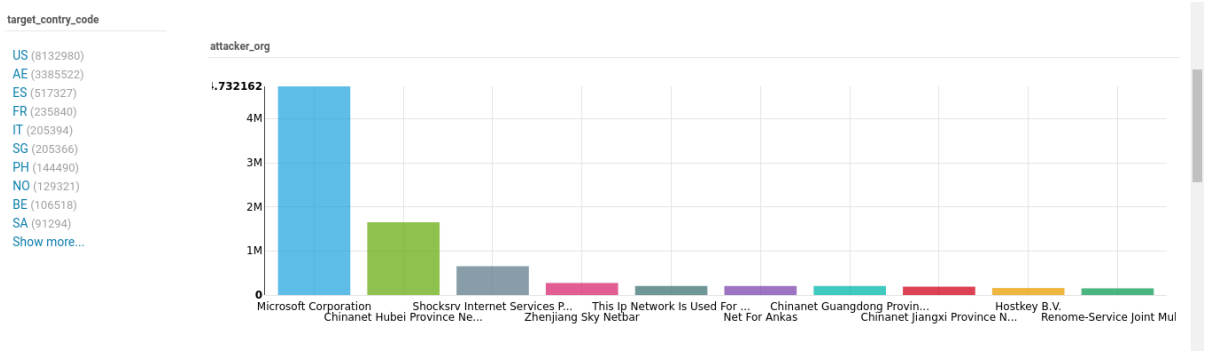


Figura 4.11: Painel de Ferramentas de Análise com detalhe para a exibição das ISPs com maior frequência.

retângulo ao lado da cidade, onde sua cor representa em qual país se encontra situada a cidade.

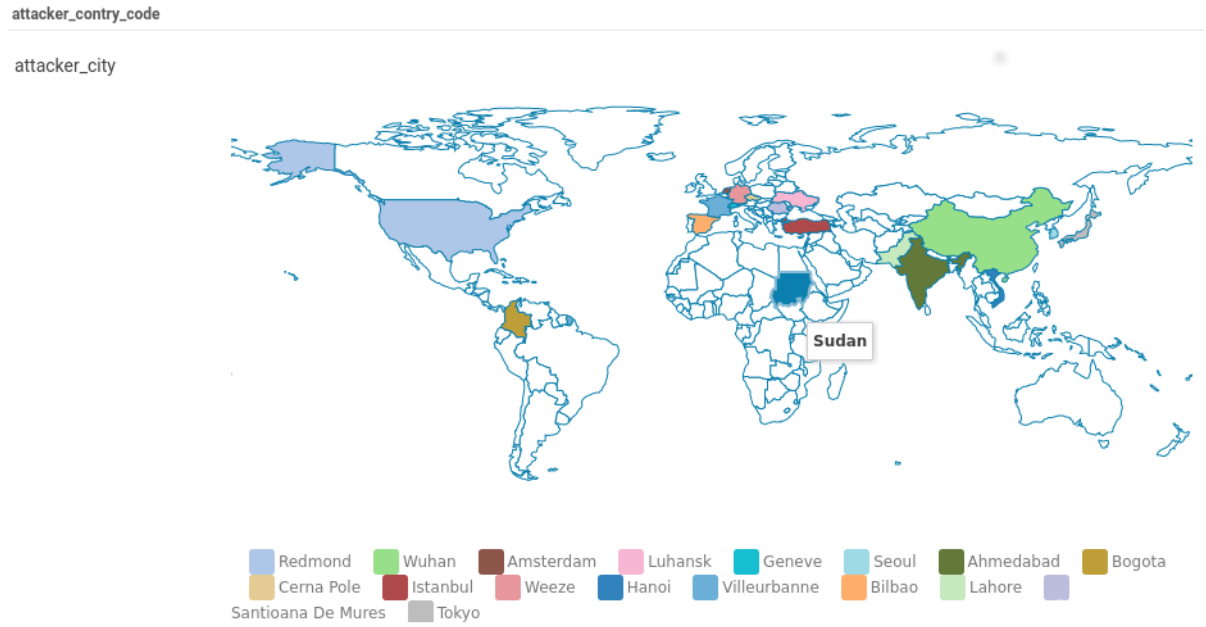


Figura 4.12: Painel de Ferramentas de Análise com detalhe para exibição de países atacantes.

A última ferramenta de análise é um painel com a possibilidade de exibição de todos os ataques com todos os seus campos ou ainda de somente aqueles campos selecionados, como pode ser visto na Figura 4.13. Denominado de *Grid Results* é possível nesse painel buscar dentro de todos os dados de ataques pode informações específicas e filtrar a apresentação destes. E ainda modificar a forma de exibição dos dados entre a em lista e a em grade.

Com toda a extensão do painel de ferramentas já explicado pode-se agora aprofundar nos resultados mais expressivos obtidos com esta nova arquitetura. Um dos primeiros

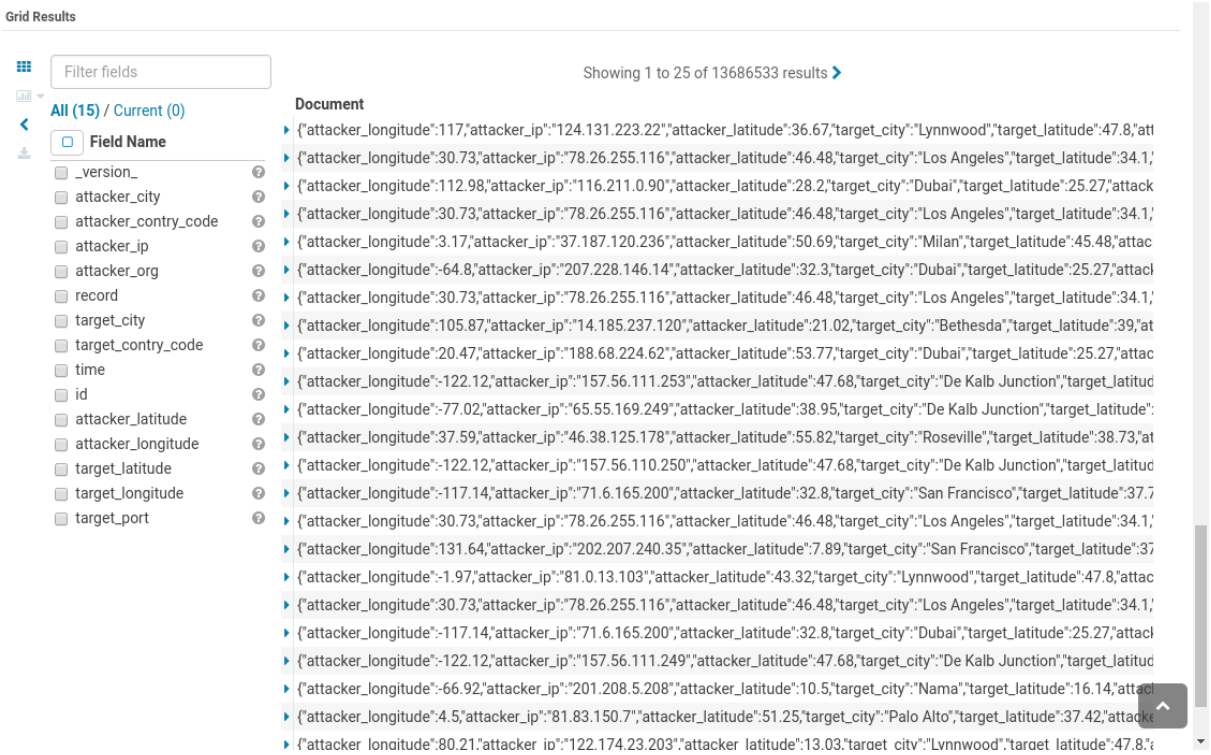


Figura 4.13: Painel de Ferramentas de Análise com detalhe para exibição das entradas do banco de dados.

resultados que pode ser obtido é a quantidade de ataques distribuído pelas portas mais atacadas, dessa forma é possível direcionar esforços a fim de se maximizar a eficiência das soluções que proteção. Outra facilidade trazida pela ferramenta diz respeito a servir como complemento para atestar origem dos ataques recebidos por determinada organização.

As portas mais atacadas bem como as quantidades de ataques em cada uma delas pode ser visto na Figura 4.14, as portas estão distribuídas segundo a quantidade de ataques recebidos da esquerda para direita e de cima para baixo.

As nove portas mais atacadas podem ser vistas na Tabela 4.2 juntamente com seus protocolos respectivos e seus responsáveis. É possível perceber já nesta tabela as duas portas que foram analisadas durante a primeira abordagem, só que agora a análise destas ocorre de forma muito mais prática e rápida.

Enquanto algumas portas da Tabela 4.2 são bastante conhecidas e utilizadas no dia a dia algumas delas ou são desconhecidas ou não são muito utilizadas por atacantes, como por exemplo as portas: 5900, 3389, 50864, 53413. A porta 5900 segundo a RFC 6143 [82] é utilizada pelo protocolo *Remote Frame Buffer* (RFB) que é aplicado no programa *Virtual Network Computing* (VNC), e tem por objetivo a transmissão de comando e telas de um ambiente gráfico.

A porta 3389 é também utilizada por um programa de envio de comandos e telas,

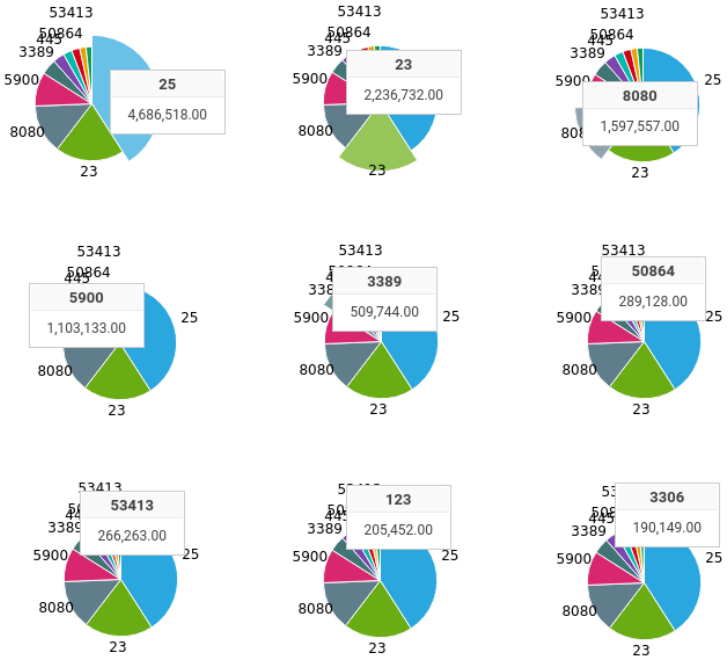


Figura 4.14: Detalhe da ferramenta de visualização de portas.

Tabela 4.2: Portas mais atacadas com descrição de protocolos e responsável

Porta	Protocolo	Reponsável
25	Simple Mail Transfer	*Internet Engineering Steering Group* (IESG)
23	Telnet	Jon Postel
8080	HTTP Alternate	Stephen Casner
5900	Remote Framebuffer	Tristan Richardson VNC
3389	MS WBT Server	Ritu Bahl
50864	Nenhum	Nenhum
53413	Nenhum	Nenhum
123	Network Time Protocol	Dave Mills
3306	MySQL	Monty

porém agora esse protocolo é descrito e mantido pela empresa Microsoft [83]. Por fim, as duas portas acima de 50.000 não possuem protocolos definidos mas como já mencionado na subseção 4.1.1 alguns dispositivos utilizam tais portas.

4.1.2.2 Aplicação de filtro de país alvo

A segunda principal utilidade desse painel de ferramentas é a possibilidade de filtragem dos ataques utilizando inúmeros parâmetros disponibilizados nos menus a esquerda. Com a distância de um clique pode-se produzir consultas plurais e diversas, permitindo desta forma a construção de resultados complexos e estruturados. Como exemplo tem-se a

Figura 4.15 que representa o painel após a escolha da filtragem por país alvo selecionando a França para análise.

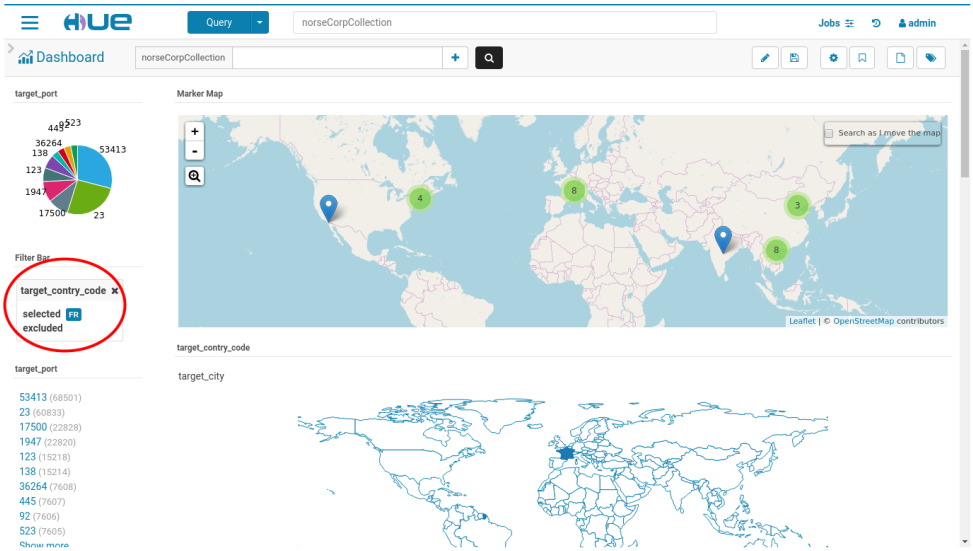


Figura 4.15: Detalhe no painel para filtragem de país alvo.

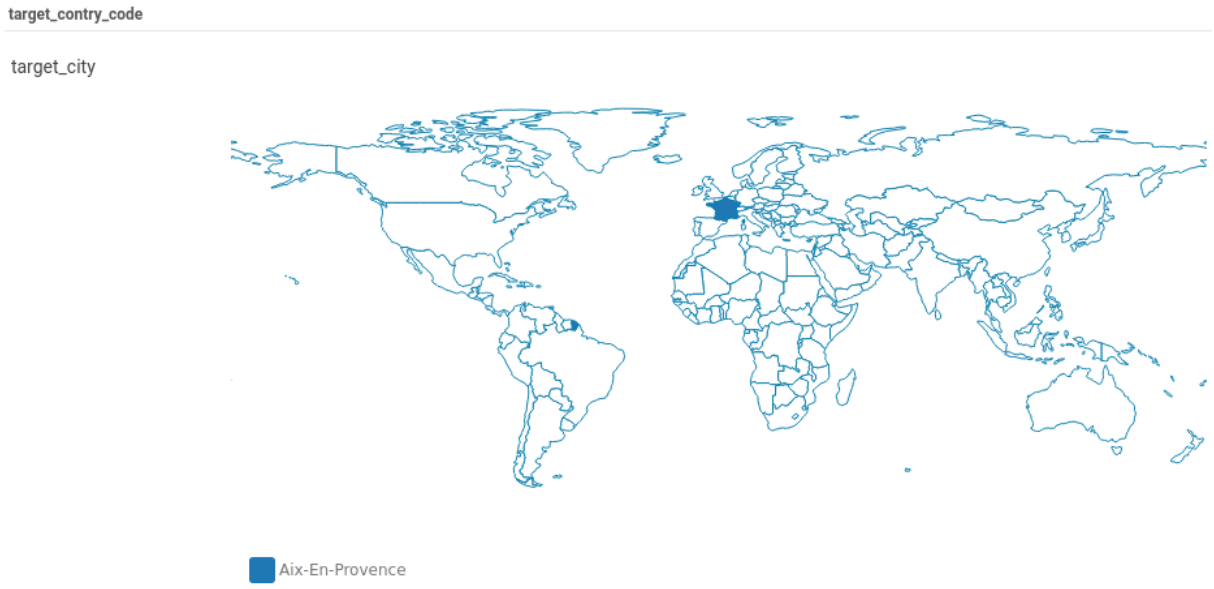
Após selecionar essa opção, todo o painel sofre modificações exibindo agora somente dados que contenham este país como alvo dos ataques. Essas modificações permitem uma completa visão de cenário tornando mais simples a análise e extração de informações para construção de um conhecimento sólido.

Os primeiros dois mapas que podem ser obtidos dizem respeito as cidades francesas que mais recebem ataques e quais os países de onde originam a maioria dos ataques, esses dois se encontram ilustrados na Figura 4.16. A Figura 4.16a mostra que nesse no caso da França somente uma cidade é alvo de ataques. Isso ocorre pois os dados coletados da empresa *NorseCorp* dizem respeito a dados capturados de seus clientes, sendo assim provavelmente a cidade de *Aix-En-Provence* é a única que possui clientes da companhia.

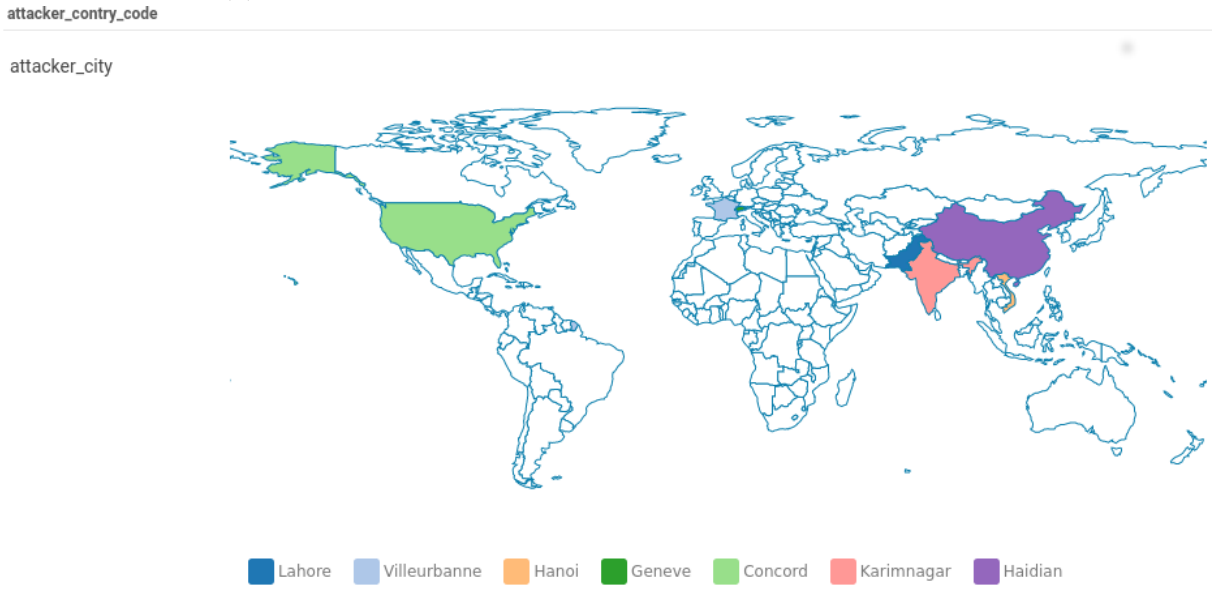
A cidade de *Aix-En-Provence* é uma cidade situada ao sul da França e possui diversos *datacenters*[4] situados em volta da cidade, em uma rápida pesquisa pela internet é possível encontrar pelo menos duas dessas instalações. Assim respaldando a informação obtida observando o mapa.

Logo em seguida, a Figura 4.16b exibe os países onde se encontram hospedadas as máquinas que enviam a maior quantidade de ataques aquele país. Para cada um dos países indicados é indicado também a cidade que detêm o maior número de ataques registrados. A relação entre a cidade e o país é novamente definida pela cor assinalada ao lado da cidade e a cor definida para cada nação.

[4]Um datacenter é uma instalação que reúne sistemas computacionais e componentes associados, como sistemas de telecomunicações, armazenamento, segurança e fornecimento de energia de backup.

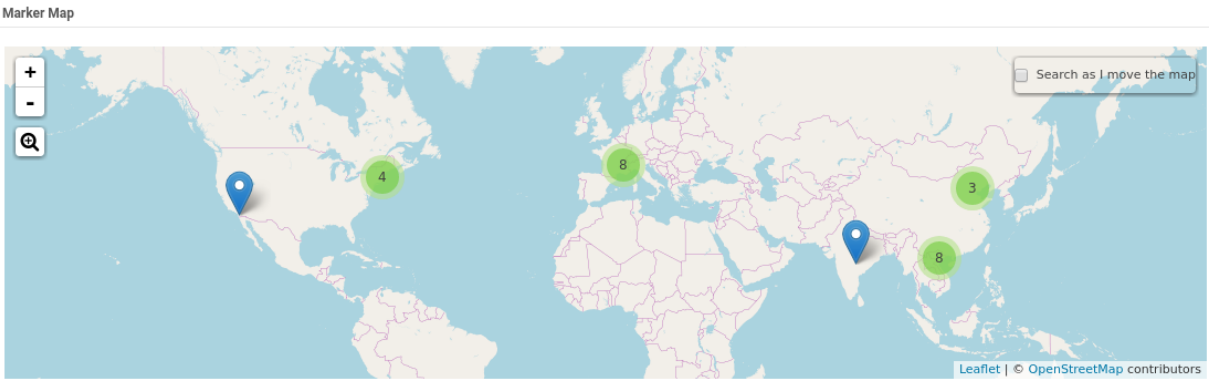


(a) Indicação das cidades com maior recebimento de ataques

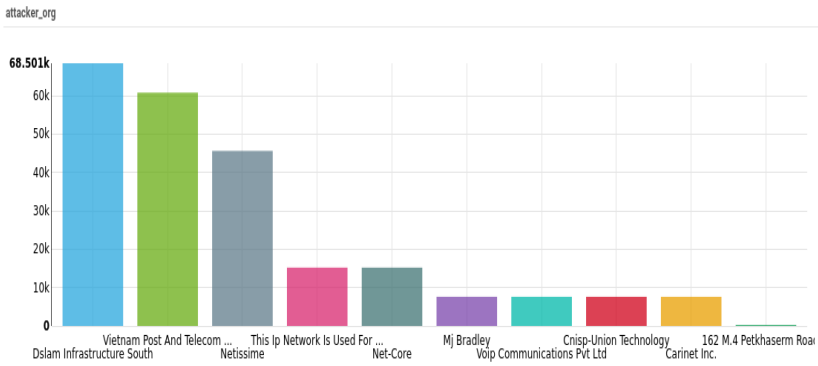


(b) Indicação dos países e cidades com maior número de atacantes

Figura 4.16: Mapas com filtro de país selecionando a França como país alvo



(a) Indicação das principais cidades atacantes



(b) Indicação das organizações responsáveis pelos IPs

(c) Indicação das portas mais atacadas

Figura 4.17: Informações resultantes após a aplicação do filtro de país com a França como país alvo

Não foram encontradas fontes confiáveis que confirmassem as informações apresentadas no gráfico representado pela Figura 4.16b, porém é válido destacar algumas informações desse. Os três primeiros atacantes dessa nação são o Paquistão, a própria França e o Vietnã. É interessante perceber que a infraestrutura de dentro do próprio país serve como plataforma para lançamento de ataques.

O restante dos resultados pode ser vista na Figura 4.17, as informações lá contidas dizem respeito a quais organizações são responsáveis pelos endereços de *IP* das máquinas que enviaram os ataques, quais portas são mais exploradas e quais regiões possuem mais atacantes. O mapa contido na Figura 4.17a tem funcionalidades específicas permitindo que o usuário escolha que posição do mapa mostra além de permitir que se dê *zoom* em regiões específicas. Outra funcionalidade possibilita filtrar as informações do painel de ferramentas enquanto se percorre o gráfico.

A Figura 4.17b faz referência as organizações assinaladas como responsáveis por administrar os endereços de rede das máquinas que lançaram os ataques sendo somente as provedoras de serviço de rede, estão listadas somente as dez maiores em ordem de-

crescente. Na Figura 4.17c as portas que aparecem mais vezes como alvo são elencadas também em ordem decrescente.

Entre as organizações que se pode identificar no gráfico estão a *Vietnam Post and Telecom, MJ Bradley* e *Union Telecom*. É importante lembrar que, essas organizações estão definidas aqui somente por serem as ISPs responsáveis pelos endereços de IP que lançaram os ataques, não sendo portanto responsáveis diretamente pelo controle dos atacantes bem como por seus danos.

Essa informação é importante pois reflete que organizações possuem o maior número de atacantes, além disso é possível a partir daí definir políticas mais rígidas quanto a acessos advindos destas ISPs. Também é possível que, as organizações entrem em contato com essas para alertar sobre atividades suspeitas e costurar acordos que beneficiem ambas as partes.

A última ferramenta é apresentada pela Figura 4.17c onde além de fornecer informação sobre as portas também permite realizar a filtragem, basta selecionar as portas desejadas que o filtro é aplicado novamente a página. Desta maneira que, se pode definir filtros diversos utilizando todos os campos recebidos durante o processo de captura.

Entre as portas francesas que mais são alvo de ataques estão as portas: 53413, 23, 17500. A primeira porta pode ser explicada devido ao fato de a França ter importado roteadores chineses vulneráveis [84], a segunda porta é alvo muito comum de ataques devido ao protocolo *Telnet* ser extremamente inseguro. Dessa forma se esta porta encontra-se aberta no dispositivo é quase certo que este pode ser subvertido.

A última porta, 17500, é definida pela IANA [85] como pertencendo ao protocolo *Dropbox LanSync Protocol* da companhia Dropbox, novamente essa informação agrega bastante valor a análise. Com ela é possível agora construir um conhecimento geral sobre as principais ameaças que se apresentam a organizações específicas permitindo aos gestores tomar medidas pró-ativas para proteção dos ativos e informações sensíveis.

4.1.2.3 Aplicação de filtro de país atacante

Assim, como é possível selecionar filtros para países que aparecem como alvo de ataques , também é apresentada a opção de filtrar os países de onde partem os ataques. O interesse de expor essa classe de filtros tem como motivação demonstrar o tamanho da variação ao se analisar dois países distintos como origem de ataques. Para ilustrar bem a diferença foram escolhidos os dois maiores países atacantes registrados na base de dados, Estados Unidos da América (US) e República Popular da China (CN).

Cada filtro foi selecionado independentemente, e os resultados das ferramentas de visualização podem ser conferidas nas Figuras 4.18 a 4.20. As primeiras duas ferramentas exibem as portas que receberam a maior quantidade de ataques, a Figura 4.18b e a

Figura 4.18a dizem respeito a informações sobre as portas mais atacadas por máquinas situadas na China. Já a Figura 4.18d e a Figura 4.18c ilustram as mesmas informações só que dos Estados Unidos da América.

Enquanto, a ferramenta exibida nas Figuras 4.18a e 4.18c tem como propósito mostrar a proporção de ataques em cada porta através de um gráfico circular, a ferramenta exposta nas Figuras 4.18b e 4.18d tem um objetivo a mais. Elas podem ser utilizadas para aplicação de um novo filtro para as informações já exibidas, da mesma forma que já havia sido mencionado na subseção 4.1.2.2.

A primeira vista fica evidente que existem várias diferenças entre os dois resultados, observando o gráfico circular é possível de imediato extrair informações claras sobre o ponto desejado. Enquanto, os ataques que partem da China tem múltiplas portas com quantidades significativas de ataques os Estados Unidos concentram os ataques em uma porta específica, sendo que as outras somente representam menos de 25% do total de ataques.

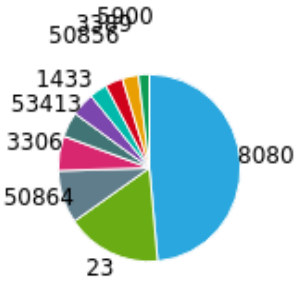
Observando a lista de portas nas Figuras 4.18b a 4.18d o usuário é capaz de perceber que a porta mais atacada pela China é a porta 8080 enquanto a partir dos Estados Unidos a porta mais utilizada é a 25.

A porta 23 figura em segundo lugar nos dois países, isso se deve ao fato já mencionado na subseção 4.1.2.2 de que essa porta pertence ao protocolo *Telnet* que é altamente inseguro. Também chama a atenção que entre as portas mais atacadas pelos americanos não há nenhuma que ultrapasse o valor de 40.000, que foi o limite definido na primeira abordagem para projeção dos gráficos Figuras 4.3 a 4.2. Mas, observando as portas chinesas há pelo menos três portas com valores superiores.

Continuando a percorrer o painel de ferramentas, duas opções de observação de mapas estáticos são mostradas, uma para representar os países e cidades que recebem mais ataques e outra para as cidades que são a origem destes ataques. Com o filtro devidamente selecionado a Figura 4.19 apresenta os mapas que contêm as cidade mais atacadas pela China e pelos Estados Unidos, respectivamente. Como já explanado na subseção 4.1.2.1, a correspondência entre as cidades e os países é representada pelas cores dos retângulos.

Há várias similaridades entre o conjunto de países atacados a partir dos Estados Unidos e da China, como por exemplo: Noruega, Estados Unidos, Rússia, França, Itália. Com os Estados Unidos acontece um fato semelhante ao ocorrido com a França, o próprio país aparece como um de seus principais atacantes. No caso dos americanos a explicação parece mais simples já que a maioria dos *datacenters* que proveem serviços de *Cloud Computing* se encontram nesse país, fato esse que se confirma ainda mais quando observamos as organizações das quais partem o maior número de ataques.

Na Figura 4.20b a organização *Microsoft Corporation* detêm quase que a totalidade

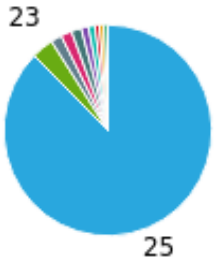


(a) Distribuição de portas atacadas a partir da China

target_port

8080 (1536689)
23 (532529)
50864 (289128)
3306 (190149)
53413 (136892)
1433 (129322)
50856 (98893)
3389 (98886)
5900 (91304)
8123 (60897)
Show more...

(b) Tabela de portas atacadas a partir da China



(c) Distribuição de portas atacadas a partir dos Estados Unidos

target_port

25 (4686518)
23 (182611)
80 (91287)
5900 (91286)
443 (83703)
138 (60868)
22 (53261)
1900 (38046)
135 (38030)
21320 (30436)
Show more...

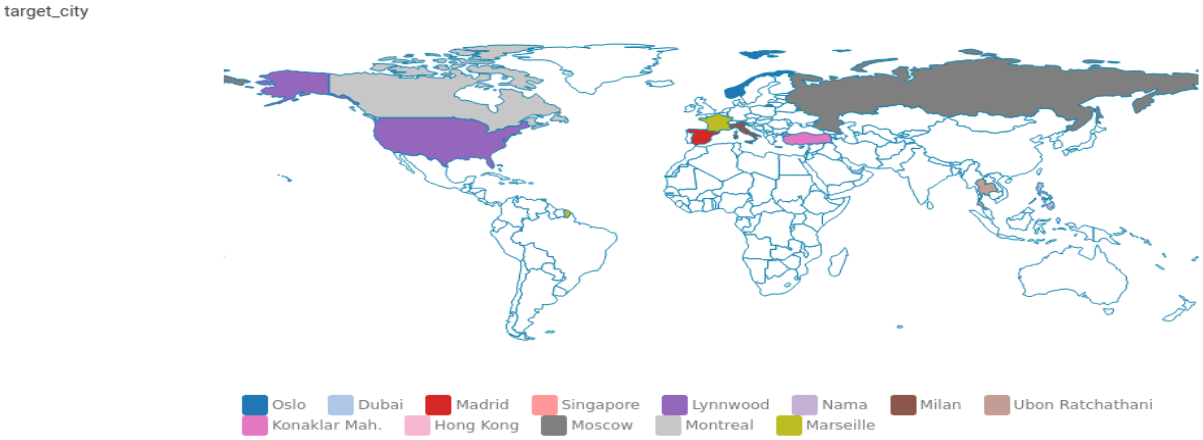
(d) Tabela de portas atacadas a partir dos Estados Unidos

Figura 4.18: Informações sobre portas atacadas a partir da China e Estados Unidos

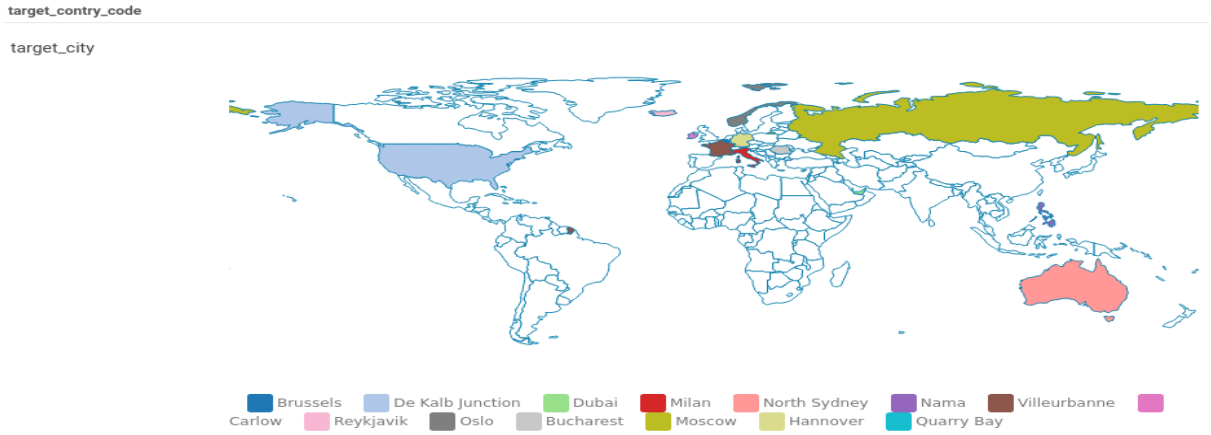
dos atacantes hospedados no país. É importante frisar que, mesmo não sendo o foco da empresa fornecer serviços de acesso à Internet, a Microsoft possui sobre seu controle várias faixas de IPs, que são utilizadas dentro de seus serviços de *Cloud*. Por esse motivo, que ela figura na lista de organizações.

Por fim, o último gráfico apresentado para essa seleção de dados é representado na Figura 4.20. Ele exibe as dez organizações que são *Internet Service Provider* (ISP) das máquinas que foram origem dos ataques capturados pela ferramenta. Os nomes de cada organização estão em ordem abaixo do gráfico e algumas vezes encontram-se deslocadas para direita.

A informação mais nítida ao se olhar para o gráfico é a da concentração dos atacantes em uma só organização para cada país, permanecendo as demais com baixíssima participação nesse ponto.

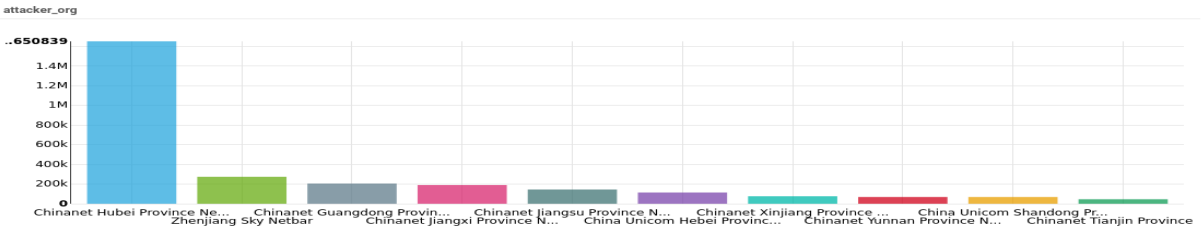


(a) Cidades mais atacadas a partir da China

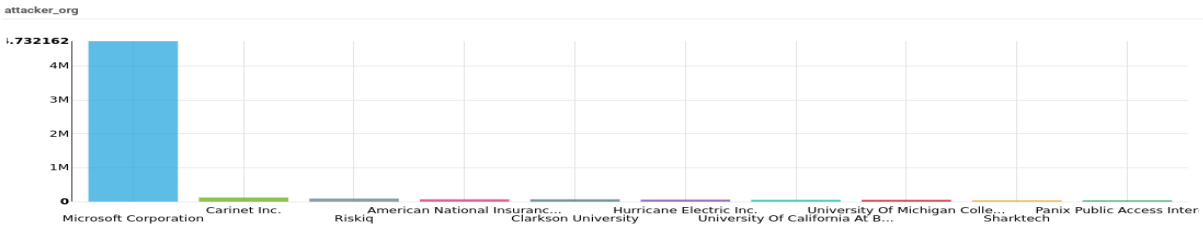


(b) Cidades mais atacadas a partir dos Estados Unidos

Figura 4.19: Cidades mais atacadas a partir da China e Estados Unidos



(a) Top 10 organizações responsáveis pelos IPs - China



(b) Top 10 organizações responsáveis pelos IPs - Estados Unidos

Figura 4.20: Top 10 organizações responsáveis pelos IPs da China e Estados Unidos

4.1.2.4 Filtragem de portas altas

Desde o início, o principal objetivo do desenvolvimento de outra arquitetura era acelerar a análise e extração de informações de um base de dados com expressiva quantidade de entradas.

Para estabelecer um bom fator de comparação, o próximo ensaio foi realizado de forma a se aproximar ao máximo da análise executada utilizando durante a primeira abordagem. Naquele momento o método K-Means foi utilizado sobre um subconjunto de dados e informações foram obtidas, porém demasiado tempo era levado para que a execução das rotinas fosse realizada.

Assim, foram definidas duas opções de filtro: ataques que tenham a porta 50864 como alvo e ataques que utilizem a porta 53413 com o mesmo objetivo. A escolha das portas se baseou em todas as informações anteriores sobre o uso de vulnerabilidades nessas porta e por conta da concentração da origem desses. Após realizar essa seleção novamente todos os dados no painel de ferramentas exibem somente os dados selecionados.

O primeiro gráfico que corrobora as informações obtidas durante a primeira abordagem encontra-se representado na Figura 4.21. Nela estão ilustradas as regiões de onde partiram os ataques que exploravam essas portas. Na Figura 4.21a somente a região leste da China é exibida, já na Figura 4.23b além desta a região da Coreia do Sul também aparece como sendo uma das principais origens.

Porém, o mapa representado na Figura 4.23b no momento da captura da imagem estava centralizado em uma região específica, portanto não exibe completamente todos os países que utilizam essa porta. Essa informação está constante na Figura 4.21, que trás os países atacantes bem como suas respectivas cidades.

Prosseguindo a fim de aumentar a precisão das observações, pode-se analisar os mapas contendo os países e as cidades que são alvos dos ataques e também as cidades específicas das quais os ataques são realizados. A Figura 4.22 ilustra os mapas que contêm a informação sobre as cidades alvo. No mapa à esquerda somente a cidade de *Lynnwood*(US) aparece como alvo de ataques que tem a porta 50864 como alvo, a direita entretanto mais cidades aparecem como alvo quando a porta 53413 é definida: a cidade de *Nama* (Japão), *Aix-En-provence* (França), *Dubai* (Emirados Árabes Unidos) e São Francisco (US).

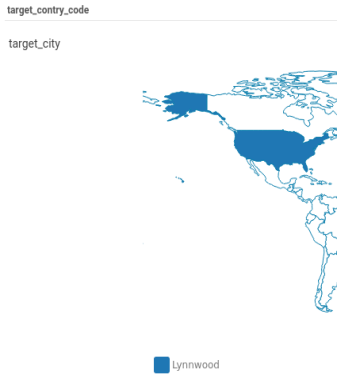
As cidades atacantes estão representadas na Figura 4.23, aqui o panorama é um pouco diferente os ataques se concentram em três nações: China, Coreia do Sul e Paquistão. Na China chama a atenção que nas duas portas escolhidas para a análise há uma cidade diferente representada, na porta 50864 a cidade de *Guanzhou* e para a porta 53413 a cidade de *Shijiazhuang* que aparece como originária dos ataques.

Como complemento à informação de regiões e cidades atacantes e alvos, há também a alternativa de se estudar o número e origem dos ataques através das ferramentas de

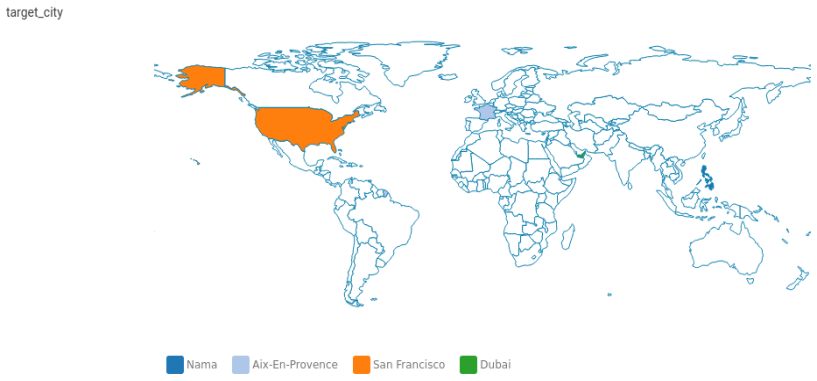
(a) Região de origem dos ataques porta 50864

(b) Região de origem dos ataques porta 53413

Figura 4.21: Regiões de origem dos ataques em portas altas



(a) País e cidade de destino dos ataques na porta 50864

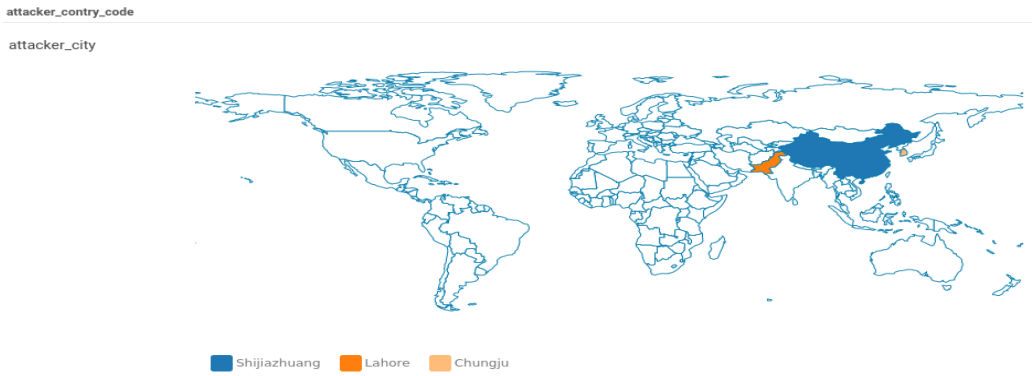


(b) País e cidade de destino dos ataques na porta 53413

Figura 4.22: Países e cidades de destino dos ataques em portas altas

filtro: *attacker_country_code* e *target_country_code*. Por vezes é mais legível acessar as informações observando-se esses campos do que as outras opções de ferramentas. A Figura 4.24 exibe essas opções de filtros para cada uma das portas.

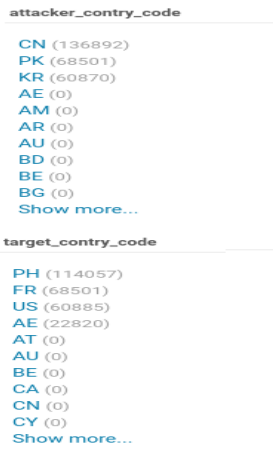
(a) País e cidade de origem dos ataques na porta 50864



(b) País e cidade de origem dos ataques na porta 53413

Figura 4.23: Países e cidades de origem dos ataques em portas altas

(a) Opções de filtro de atacante e alvo com quantidade de ataques porta 50864

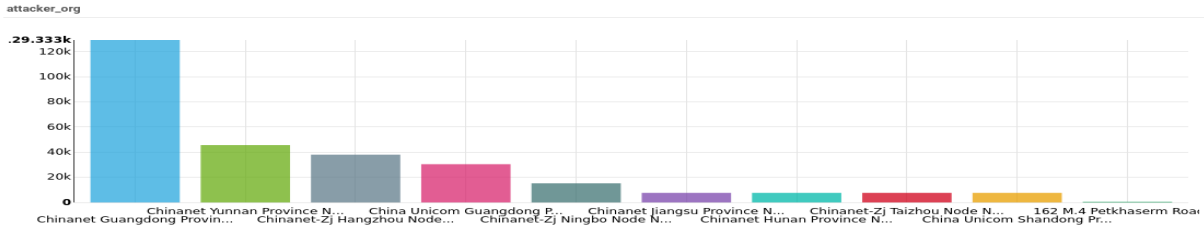


(b) Opções de filtro de atacante e alvo com quantidade de ataques porta 53413

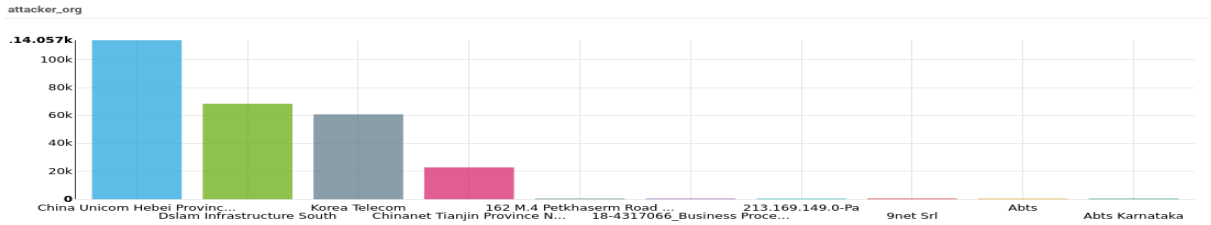
Figura 4.24: Opções de filtros para atacante e alvo para análise de portas altas

Por último, as organizações que são a *Internet Service Provider* (ISP) das máquinas atacantes estão elencadas no gráfico contido na Figura 4.25.

A segunda base de dados capturada também trouxe importantes informações, dada as



(a) Top 10 organizações responsáveis pelos IPs com ataques na porta 50864

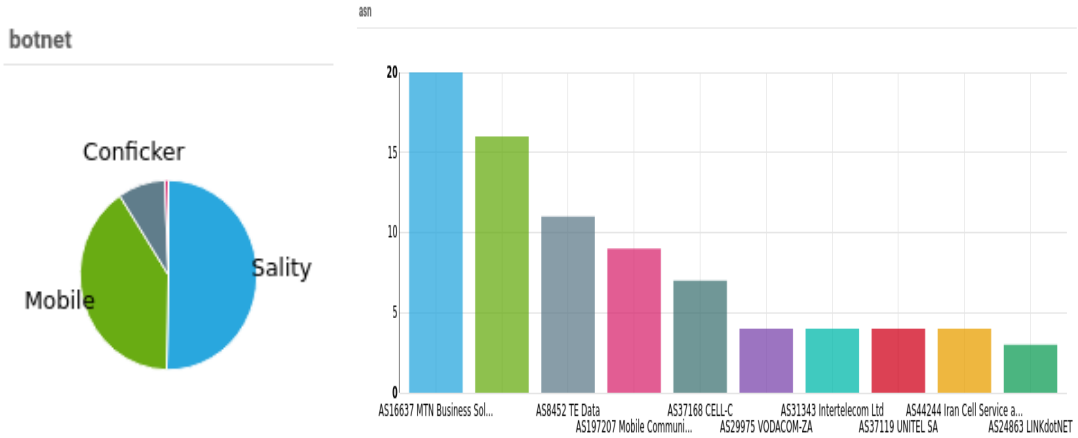


(b) Top 10 organizações responsáveis pelos IPs com ataques na porta 53413

Figura 4.25: Organizações responsáveis pelos IPs com ataques em portas altas

diferenças entre os dados capturados há aqui uma outra perspectiva sobre os ataques. Na Figura 4.26 é possível perceber as *botnets* mais ativas bem como as *Autonomous system Network* (ASN) das quais partem a maior parte dos ataques.

De fato, as informações sobre as *botnets* encontram embasamento, durante uma consulta as principais empresas de tecnologia de segurança em rede é possível constatar o aumento do uso de tais ferramentas e suas variantes em ataques de rede [86, 87].



(a) Distribuição *botnets* (b) Distribuição ASNs

Figura 4.26: Resultados base de dados Looking Glass Cyber

Outra informação pertinente foi a distribuição dos atacantes por país como pode ser visto na Figura 4.27. É possível perceber que vários países estão presentes tanto nesta observação quanto na obtida através da base de dados da NorseCorp, como por exemplo, China, Estados Unidos e Rússia. Entretanto, também há alguns países que não apre-

sentavam quantidades significativas na primeira base e que aqui representam um grande valor, como por exemplo África do Sul, Marrocos e Angola.

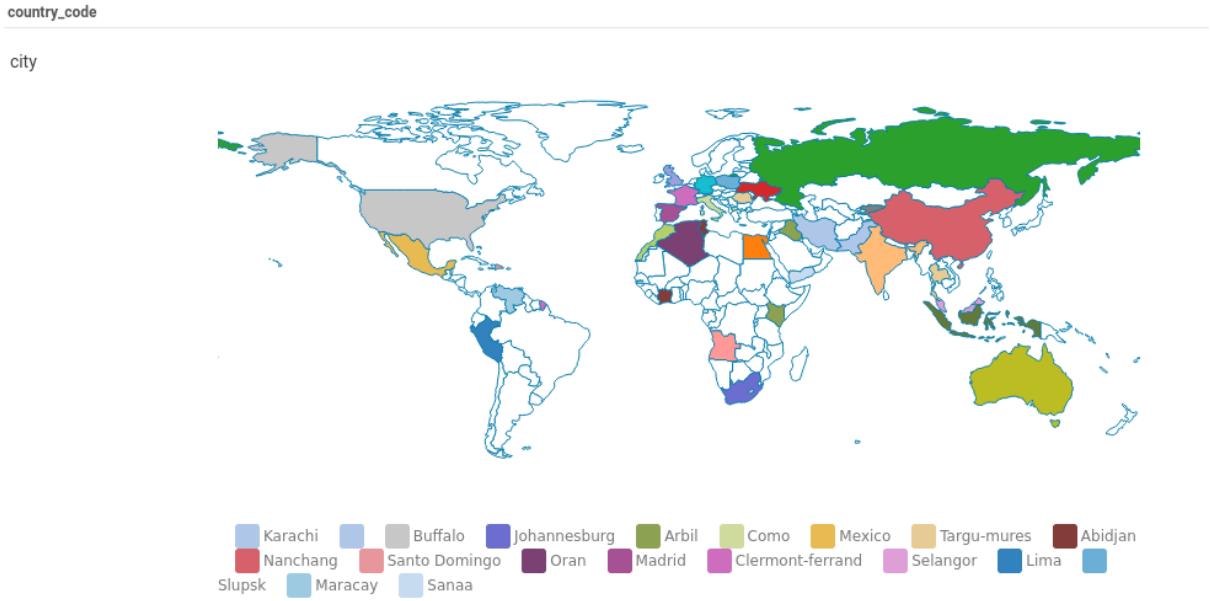


Figura 4.27: País e cidade com maior número de atacantes.

4.2 Análise

No início do desenvolvimento deste trabalho, a ideia era aplicar métodos para tratamento e extração de informações, com o objetivo de enriquecê-los e gerar informação pertinente sobre inteligência de ameaças. Esse foi o proposito durante o desenvolvimento da primeira abordagem, o foco não era a arquitetura mas sim o processamento em si. Por esse motivo linguagens e programas simples com curvas de aprendizado suave foram escolhidos inicialmente.

Porém, logo ficou claro que essa decisão não foi adequada, dois pontos principais causavam muitos problemas. O primeiro deles era que o tempo de processamento crescia de forma vertiginosa conforme o número de ataques capturados aumentava, o segundo estava relacionado a que qualquer nova escolha de método ou de um novo subconjunto para análise gerava um novo trabalho de desenvolvimento.

Com cerca de dez milhões de dados armazenados na base o tempo consumido para geração de um gráfico simples era de mais de cinco minutos, se após essa primeira projeção se quisessem modificar o intervalo de dados ou se escolher novos domínios era necessário retornar ao desenvolvimento do programa ou então passar novos argumento e aguardar novamente que o programa executasse.

Nesse ponto ainda não era possível afirmar qual melhor método para o processamento dos dados e nem quais combinações destes gerariam o melhor resultado, assim foi necessário realizar uma série de testes procurando encontrar a melhor forma de extrair informações daquela imensa base. Mas a primeira abordagem cobrava um preço caro para realização destes testes, o reduzido número de resultados apresentados na subseção 4.1.1 se deve a isso.

Dessa forma, uma mudança completa de plano teve que ser realizada. Agora o foco não era mais em quais os melhores métodos ou técnicas para extrair informações mas sim em como fazer com que o processamento desses dados ocorresse de forma fácil e ágil, para assim oferecer liberdade e permitir a produção de resultados mais completos.

A escolha por várias fases de planejamento e estruturação, buscando não cometer os mesmos erros do passado a concepção agora se apoiava em construir um sistema capaz de lidar com uma base de dados que tende sempre a crescer, sem que para isso se sacrifique o tempo de processamento e a facilidade de uso.

Assim, foi proposta a segunda abordagem, que utilizasse o paradigma de *Big Data* sobre um *cluster* de máquinas. A suíte de programas do Apache Hadoop em conjunto com outras ferramentas foi aplicada com o gerenciamento da infraestrutura nas mãos da aplicação Cloudera o sistema agora era de simples manutenção e utilização, versátil, elástico e escalável.

Os ganhos foram imensos, o processamento dos dados agora ocorria à distância de um clique. A escolha de novos dados para projeção são executados de forma muito rápida carregando as novas informações na tela de forma que seu entendimento seja fácil. Com o armazenamento dos dados em um sistema de tabelas distribuída (HBase) o posterior processamento dos dados também apresenta expressivo ganho, se a quantidade de dados aumentar basta inserir novos nós no *cluster* que a capacidade total de processamento e armazenamento cresce quase que instantaneamente.

Todas as ferramentas da fundação *Apache* para aplicação com o paradigma de *Big Data* foram desenvolvidas de forma que estas ferramentas possam ter seu processamento realizado de forma distribuída. Além do processamento o armazenamento de dados também foi escalonado de forma que os dados estivessem dispersos dentro da infraestrutura, desta forma as limitações para leitura e escrita de arquivos poderia ser reduzida.

Outro ponto a favor de uma arquitetura desenvolvida utilizando essas ferramentas reside no fato de que grandes empresas de computação na nuvem já oferecem essas como serviço, como por exemplo a empresa Amazon que em sua gama de produtos possuí os programas HBase, Apache Hadoop, Apache Spark entre outros. E com custos cobrados por utilização o que torna viável sua utilização.

Entretanto, nem todos os pontos da primeira abordagem são ruins, o programa de

captura de dados utilizado nesta foi reaproveitado devido a sua eficiência. Algumas modificações foram inseridas, porém sua forma geral não precisou ser alterada. O banco de dados utilizado na primeira abordagem possuía um bom desempenho quando o assunto é permitir a inserção de dados, porém como já citado há alguns problemas quanto ao consumo de memória deste.

Mesmo apresentando um bom desempenho, o banco de dados utilizado não se aproxima da capacidade projetada da ferramenta Flume, essa por sua vez possuí suporte a diversas fontes de dados que podem ser escaladas de forma hierárquica e fazer uso de um sistema distribuído para captura e armazenamento dos dados.

Enquanto utilizando o banco de dados da primeira abordagem o gargalo do sistema era quanto de memória estava disponível, no Flume o único gargalo é quantas máquinas podem ser acopladas ao *cluster*. Outro fator de superioridade dessa ferramenta é a possibilidade nativa de distribuição de suas instâncias, o que não pode ser feito de forma simples com o banco de dados *MongoDB*.

Capítulo 5

Conclusão

As possibilidades abertas pela nova abordagem são inúmeras, de forma que não foi possível explorar todas no tempo disponível para conclusão deste trabalho. Ficando no final o foco voltado para o desenvolvimento da arquitetura e seu correto funcionamento, mesmo assim ainda foi possível fazer uso de algumas ferramentas que já demonstram as perspectivas de sua utilização.

A apresentação das informações de forma visual com simplicidade e ilustrações de fácil compreensão permitem que um usuário, mesmo que desprovido de informações sobre como aqueles dados estão sendo processados, possa entender e interagir com esses. Isso permite que padrões sejam observados e que o usuário com seu conhecimento pregresso ou através de outras fontes possa enriquecer os dados gerando informações e as validando.

Assim, aquela massa de dados contendo escassas informações pode ser convertida, através de diversas técnicas de processamento, em informações que agregam diversos conceitos e que geram uma visão completamente nova do que se julgava ter pleno conhecimento.

A combinação dos dados apresentadas no painel de ferramentas trazem as principais informações que se necessita para avaliação, auditoria, rastreabilidade de ataques e para o planejamento de uma política sólida de segurança da informação. É possível avaliar desde quais portas recebem mais ataques até qual país tem maior participação nos ataques a determinada organização.

Os ganhos para organizações privadas e públicas ocorre quando a interpretação das informações gera um efeito ativo sobre a segurança da informação, agora não é preciso esperar que as ameaças infrinjam dados aos ativos organizacionais. As ações podem ser tomadas pró-ativamente evitando assim gastos não previstos e exposição de informações sensíveis.

Outra utilização da ferramenta é a de conseguir representar as mudanças ao longo do tempo, assim o comportamento geral das informações analisadas pode ser comparada para se definir uma tendência ou várias delas. Projetando diversos cenários para o futuro

que podem alimentar o Plano Diretor de Tecnologia da Informação (PDTI) com os níveis de ameaça esperados.

Para um país o uso da ferramenta parece ser mais vantajosa, pois permite que seja utilizado em um ambiente de inteligência de ameaças em nível global permitindo um correto posicionamento da nação de forma muito mais enfática e convicta.

5.1 Trabalhos Futuros

Processamento utilizando clusterização com Apache Spark - O *framework* Apache Spark conta com diversas implementações de métodos de extração de dados supervisionados e não supervisionados como o K-Means. Trabalhos futuros podem fazer uso deste para divisão em *cluster* dos dados persistidos na arquitetura, pra facilidade de recuperação que esses dados foram armazenados no sistema HBase. O Spark também possui uma biblioteca para aprendizado de máquina, que representa um boa perspectiva com a utilização dos dados armazenados para treinamento de uma rede que possa interpretar os dados de forma autônoma.

Captura de diversas fontes - Quanto mais fontes forem inseridas mais robusto e confiável ficará o sistema. Existem diversas fontes abertas na internet que enviam dados por meio do protocolo HTTP, a integração dessas novas fontes ao sistema não deve apresentar grandes problemas já que o desenvolvimento da arquitetura buscou tornar simples o acoplamento dessas novas fontes. Além disso com novas fontes de dados ferramentas de comparação para afirmar a veracidade dos dados podem ser implementadas.

Comparação entre diversas fontes não homogêneas para corroboração de dados - Para enriquecimento dos dados é necessário que um ser humano observe as informações e utilizando de seu conhecimento prévio ou de pesquisas mais aprofundadas consiga confirmar os dados e indicar as fontes e vulnerabilidades daqueles ataques. A inserção de um meio automático para o enriquecimento desses dados é fundamental para que seja possível fornecer informações verdadeiras em um volume considerável

Geração de relatórios automáticos - Depois de inseridas as ferramentas para enriquecimento automático a geração de relatórios automáticos é um trabalho futuro interessante, tendo inclinação para prestação de um serviço de inteligência de ataques em rede e ameaças prováveis.

Referências

[1] *Teoria da Informação*. http://www.ib.usp.br/~rpavao/Teoria_da_Informacao.pdf. 3

[2] Shannon, Claude E: *A mathematical theory of communication, part i, part ii*. Bell Syst. Tech. J., 27:623–656, 1948. 4

[3] Shannon, Claude E: *Communication theory of secrecy systems*. Bell Labs Technical Journal, 28(4):656–715, 1949. 4

[4] Gleick, James: *A informação: uma história, uma teoria, uma enxurrada*. Editora Companhia das Letras, 2013. 4

[5] Valdemar W.Setzer: *Data, Information, Knowledge and Competence*. 3rd CONTECSI (International Conference on Information Sysytems), jun 2006. 4

[6] Paulo, Planez: *Um pouco de História para entender os sistemas de informação*, 2015. https://www.tiespecialistas.com.br/2015/10/um-pouco-de-historia-para-entender-os-sistemas-de-informacao/, acesso em 2017-12-14. 5

[7] Amaral, Antônio Eugênio Maia: *1000 anos antes de Gutenberg*. Cadernos BAD, 2, 2003. https://scholar.google.com.br/scholar?hl=pt-BR&as_sdt=0%2C5&q=bi+sheng+prensa&btnG=. 5

[8] Ray, Russell: *The "Wicked" Bibles*. Theology Today, 37(3):360–363, 1980. http://journals.sagepub.com/doi/10.1177/004057368003700311#articleCitationDownloadContainer. 5

[9] Mantoux, Paul: *A revolução industrial no século XVIII: estudo sobre os primórdios da grande indústria moderna na Inglaterra.* Editora UNESP, São Paulo, 1988, ISBN 8527100452. https://www.passeidireto.com/disciplina/teorias-da-administracao-i?type=6&materialid=3054240. 6

[10] Martins, Eliseu: *Contabilidade de Custos*. Atlas, São Paulo, 9ª edição, 2003, ISBN 85-224-3360-7. https://s3.amazonaws.com/academia.edu.documents/39402522/contabilidade_de_custos.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1513353921&Signature=P0JfmTjU3faL6ZK7vxXe7a0IjSI%3D&response-content-disposition=inline%3Bfilename%3DEliseu_Martins.pdf. 6

[11] PAMPLONA, Edson de O.: *A Contabilidade Gerencial*, 1998. http://www.iem.efei.br/edson/, acesso em 2017-12-13. 6

[12] Briggs, Charles Frederick e Augustus Maverick: *The Story of the Telegraph, and a History of the Great Atlantic Cable, Etc.[With a Plate and a Map.]*. Rudd & Carleton, 1858. 6

[13] Kardex Remstar: *History - Kardex Remstar*. http://www.kardex-remstar.com/en/kardex-remstar-about-us/history.html, acesso em 2017-12-15. 6

[14] Moura, Reinaldo Aparecido: *Kanban: a simplicidade do controle da produção*. IMAM, 1994. 6

[15] Hat, Red: *A Brief History of Cryptography - Red Hat Customer Portal*, 2013. https://access.redhat.com/blogs/766093/posts/1976023, acesso em 2017-12-16. 7

[16] H. Salus, Peter: *Net Insecurity: Then and Now (1969-1998)*, 1998. http://www.sane.nl/events/sane98/aftermath/salus.html, acesso em 2017-12-16. 8

[17] K. C., LAUDON e LAUDON J. P.: *Sistemas de Informações Gerenciais*. Pearson Prentice Hall, São Paulo, 7ª edição, 2007. https://pt.wikipedia.org/wiki/Mainframe#cite_note-:0-1. 8

[18] Hauben, Michael: *History of arpanet*. History of ARPANET. Available at http://www. dei. isep. ipp. pt/˜ acc/docs/arpa. html (14 October 2012), 2007. 8

[19] Whitman, Michael E. e Herbert J Mattord: *Principles of Information Securit*. Cengage Learning, 2011. 8, 9, 10

[20] Ware, Willis H: *Security controls for computer systems. report of defense science board task force on computer security*. Relatório Técnico, RAND CORP SANTA MONICA CA, 1979. 8

[21] Perrin, Chad: *The CIA triad*. Dostopno na: http://www. techrepublic. com/blog/security/the-cia-triad/488, 2008. 9

[22] ABNT, NBRISO: *IEC 27.002: 2013 (antiga NBR ISO/IEC 17799: 2005)-Código de Prática para a Gestão da Segurança da Informação*, 2013. 9

[23] Liptak, Deborah: *Information Warfare: A Glossary of Enmity*. Searcher, 19(4):22–33, 2011, ISSN 1070-4795. http://search.proquest.com/docview/881453961/. 10

[24] Bill, Brenner: *Threat Advisory: High-Risk Zeus Crimeware Kit - The Akamai Blog*, 2014. https://blogs.akamai.com/2014/06/threat-advisory-high-risk-zeus-crimeware-kit.html, acesso em 2018-01-01. 12

[25] Akamai: *What is Ransomware?* https://www.akamai.com/us/en/resources/what-is-ransomware.jsp, acesso em 2018-01-01. 12

[26] Akamai e Seaman Chad: *Threat Advisory: Mirai Botnet*. Relatório Técnico, 2017. https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/akamai-mirai-botnet-threat-advisory.pdf. 12

[27] Akamai: *State of the Internet report for the second quarter of 2013*, 2014. https://blogs.akamai.com/2013/10/indonesia-attack-traffic-tops-list-port-445-no-longer-main-launching-pad.html, acesso em 2017-12-13. 12, 69

[28] Magnuson, Stew: *CYBER WAR*. National Defense, 90(627):30–31, feb 2006, ISSN 00921491. http://search.proquest.com/docview/213300519/?pq-origsite=primo. 13

[29] Rhett, Jones: *A OTAN está considerando o NotPetya um potencial ato de guerra*, 2017. http://gizmodo.uol.com.br/otan-petya-potencial-ato-guerra/, acesso em 2017-12-18. 13

[30] *U.S. Fleet Cyber Command (FCC)/U.S. TENTH Fleet (C10F)*. http://www.public.navy.mil/fcc-c10f/Pages/home.aspx, acesso em 2017-12-18. 13

[31] DAVID, KUSHNER: *The Real Story of Stuxnet*, 2013. https://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet, acesso em 2017-12-18. 13

[32] Nakashima, Ellen e Joby Warrick: *Stuxnet was work of us and israeli experts, officials say*. Washington Post, 2, 2012. 13

[33] Broad, William J, John Markoff e David E Sanger: *Israeli test on worm called crucial in iran nuclear delay*. New York Times, 15:2011, 2011. 13

[34] White, Tom: *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012. 13, 31, 34, 35, 36, 37, 38, 39

[35] David, Reinsel, Gantz John e Rydning John: *Data Age 2025: The Evolution of Data to Life-Critical*. Relatório Técnico, IDC, 2017. https://www.seagate.com/files/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf. 13

[36] IBM, Zikopoulos Paul e Eaton Chris: *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill Osborne Media, 1ª edição, 2011, ISBN 0071790535 9780071790536. https://dl-acm-org.ez54.periodicos.capes.gov.br/citation.cfm?id=2132803. 14

[37] I., Fette, Inc. Google, Melnikov A. e Ltd. Isode: *The WebSocket Protocol*. Relatório Técnico, IETF, 2011. https://tools.ietf.org/html/rfc6455. 17, 22

[38] W3C: *HTML5 Introduction*. https://www.w3schools.com/html/html5{_}intro.asp, acesso em 2018-01-01. 17

[39] Leach, Paul J., Tim Berners-Lee, Jeffrey C. Mogul, Larry Masinter, Roy T. Fielding e James Gettys: *Hypertext Transfer Protocol – HTTP/1.1*. IETF, página 176, 1999. https://tools.ietf.org/html/rfc2616. 18

[40] Kozierok, Charles M: *The TCP/IP guide: a comprehensive, illustrated Internet protocols reference*. No Starch Press, 2005. 19

[41] Tukey, J W: *Exploratory Data Analysis*. Addison-Wesley series in behavioral science. Addison-Wesley Publishing Company, 1977, ISBN 9780201076165. https://books.google.com.br/books?id=UT9dAAAAIAAJ. 23

[42] Duda, R O, P E Hart e D G Stork: *Pattern classification*. Pattern Classification and Scene Analysis: Pattern Classification. Wiley, 2001, ISBN 9780471056690. https://books.google.com.br/books?id=YoxQAAAAMAAJ. 23

[43] Jain, Anil K: *Data clustering: 50 years beyond K-means*. Pattern Recognition Letters, 31(8):651–666, 2010, ISSN 0167-8655. 23

[44] Steinhaus, H: *Sur la division des corp materiels en parties*. Bull. Acad. Polon. Sci, 1:801–804, 1956. 24

[45] Lloyd, S: *Least squares quantization in PCM*. Information Theory, IEEE Transactions on, 28(2):129–137, mar 1982, ISSN 0018-9448. 24, 51

[46] Ball, Geoffrey H e David J Hall: *ISODATA, a novel method of data analysis and pattern classification*. Relatório Técnico, Stanford research inst Menlo Park CA, 1965. 24

[47] MacQueen, James e Others: *Some methods for classification and analysis of multivariate observations*. Em *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, páginas 281–297, 1967. 24

[48] Drineas, P, A Frieze, R Kannan, S Vempala e V Vinay: *Clustering Large Graphs via the Singular Value Decomposition*. Machine Learning, 56(1):9–33, jul 2004, ISSN 0885-6125. 24

[49] Meilă, Marina: *The uniqueness of a good optimum for k-means*. Em *Proceedings of the 23rd international conference on Machine learning*, páginas 625–632, 2006. 24

[50] SciPy: *Scientific Computing Tools for Python — SciPy*. https://www.scipy.org/about.html, acesso em 2017-12-19. 26

[51] NumPy: *NumPy*. http://www.numpy.org/, acesso em 2017-12-19. 26

[52] Cloudera, Inc.: *Cloudera Manager: Hadoop Administration tool*. https://www.cloudera.com/products/product-components/cloudera-manager.html, acesso em 2018-01-01. 28

[53] Cloudera, Inc.: *Cloudera company background information, management & board*. https://www.cloudera.com/more/about.html, acesso em 2018-01-01. 28

[54] Philip, Zeyliger: *How Does Cloudera Manager Work? – Cloudera Engineering Blog*, 2013. http://blog.cloudera.com/blog/2013/07/how-does-cloudera-manager-work/, acesso em 2017-12-19. 29

[55] Borthakur, Dhruba *et al.*: *Hdfs architecture guide*. Hadoop Apache Project, 53, 2008. 31

[56] Code, Emphasis: *hdfs-arch.jpg*. `https://codemphasis.files.wordpress.com/2012/09/hdfs-arch.jpg`, acesso em 2017-12-20. 32

[57] Dean, Jeffrey e Sanjay Ghemawat: *MapReduce: simplified data processing on large clusters*. Communications of the ACM, 51(1):107–113, 2008. 32

[58] Amazon, AWS: *MapReduce.png*. `http://mm-tom.s3.amazonaws.com/blog/MapReduce.png`, acesso em 2017-12-20. 33

[59] Vavilapalli, Vinod Kumar, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth *et al.*: *Apache hadoop yarn: Yet another resource negotiator*. Em *Proceedings of the 4th annual Symposium on Cloud Computing*, página 5. ACM, 2013. 34

[60] Vora, Mehul Nalin: *Hadoop-hbase for large-scale data*. Em *Computer science and network technology (ICCSNT), 2011 international conference on*, volume 1, páginas 601–605. IEEE, 2011. 35

[61] Apache Software Foundation: *Apache ZooKeeper - Home*. `http://zookeeper.apache.org/`, acesso em 2017-12-19. 35

[62] Fw1-aus1): *ZooKeeper Overview*, 2010. `https://wiki.apache.org/hadoop/ZooKeeper/ProjectDescription`, acesso em 2017-12-19. 37

[63] Zaharia, Matei, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin *et al.*: *Apache spark: A unified engine for big data processing*. Communications of the ACM, 59(11):56–65, 2016. 37

[64] Foudation, Apache: *Apache Flume*, 2017. `https://flume.apache.org/`, acesso em 2017-12-20. 38

[65] Apache Software Foundation: *DevGuide_image00.png*. `http://flume.apache.org/_images/DevGuide_image00.png`, acesso em 2017-12-20. 38

[66] Apache Software Foundation: *Flume image 01*. `https://flume.apache.org/_images/UserGuide_image01.png`, acesso em 2017-12-20. 42

[67] Apache Software Foundation: *Flume image 02*. `https://flume.apache.org/_images/UserGuide_image02.png`, acesso em 2017-12-20. 42

[68] Cloudera Inc.: *Flume image 03*. `https://archive.cloudera.com/cdh5/cdh/5/flume-ng/_images/UserGuide_image03.png`, acesso em 2017-12-20. 42

[69] Foudation, Apache: *Apache Kafka*, 2017. `https://kafka.apache.org/`, acesso em 2017-12-20. 39

[70] Evan, Mouzakitis: *Monitoring Kafka performance metrics*, 2016. https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/, acesso em 2017-12-20. 39

[71] Foudation, Apache: *Apache SOLR*, 2012. http://lucene.apache.org/solr/, acesso em 2017-12-20. 40

[72] Marco, Antonio Rocha: *Solr: Indexação e Buscas de Alta Performance usando Software Open Source | MATERA Systems*, 2012. http://www.matera.com/br/2012/08/20/solr-indexacao-e-buscas-de-alta-performance-usando-software-open-source/, acesso em 2017-12-20. 40

[73] Apache Software Foundation: *PublicServers - Solr Wiki*. https://wiki.apache.org/solr/PublicServers, acesso em 2017-12-20. 40

[74] Hue: *Hue - Hadoop User Experience - The Apache Hadoop UI | Hue is a Web application for querying and visualizing data by interacting with Apache Hadoop*, 2014. http://gethue.com/. 43

[75] Maps, of World: *world-map-with-latitude-and-longitude.jpg*. https://www.mapsofworld.com/images2008/world-map-with-latitude-and-longitude.jpg, acesso em 2017-12-20. 63

[76] Gman_beeman: *My Router is Showing a Bunch of IPs Attacking Me*. https://facepunch.com/showthread.php?t=1037987. 64

[77] Jill Scharr: *Possible Backdoor Found in Chinese-Made Routers*, 2014. https://www.tomsguide.com/us/chinese-router-backdoor,news-19398.html, acesso em 2017-12-13. 64

[78] Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil, CERT.br: *Cartilha de Segurança para Internet*. Comitê Gestor da Internet no Brasil, São Paulo, 2ª edição, 2012, ISBN 978-85-60062-54-6. https://cartilha.cert.br/livro/cartilha-seguranca-internet.pdf. 64, 65

[79] Eduard Kovacs: *Easily Exploitable Vulnerability Found in Netis Routers*, 2014. http://www.securityweek.com/easily-exploitable-vulnerability-found-netis-routers, acesso em 2017-12-13. 64

[80] Chad Seaman: *Threat Advisory: Mirai Botnet*. Relatório Técnico, Akamai, 2016. https://www.akamai.com/us/en/about/our-thinking/threat-advisories/akamai-mirai-botnet-threat-advisory.jsp. 65

[81] DAN GOODIN: *Newly discovered router flaw being hammered by in-the-wild attacks*, 2016. https://arstechnica.com/information-technology/2016/11/notorious-iot-botnets-weaponize-new-flaw-found-in-millions-of-home-routers/, acesso em 2017-12-13. 67

[82] T., Richardson e Levine J.: *The Remote Framebuffer Protocol*, 2011. https://tools.ietf.org/html/rfc6143, acesso em 2017-12-13. 73

[83] Microsoft Corporation: *Remote Desktop Protocol: Basic Connectivity and Graphics Remoting*, 2017. https://msdn.microsoft.com/en-us/library/cc240445.aspx, acesso em 2017-12-13. 74

[84] System, Netis: *Netis Partners*, 2017. http://www.netis-systems.com/Partners/index/st/1/ar/4.html, acesso em 2017-12-20. 78

[85] Touch, Joe, Eliot Lear, Allison Mankin, Markku Kojo, Kumiko Ono, Martin Stiemerling, Lars Eggert, Alexey Melnikov, Wes Eddy, Alexander Zimmermann, Brian Trammell, Jana Iyengar, Allison Mankin andMichael Tuexen, Eddie Kohler e Yoshifumi Nishida: *Service Name and Transport Protocol Port Number Registry*, 2017. https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml, acesso em 2017-12-13. 78

[86] Falliere, Nicolas: *The sality botnet*. http://www.symantec.com/connect/blogs/sality-botnet, maio 2010. 85

[87] Avi, Aminov: *Spotlight on Malware DGA Communication Technique*, 2017. https://blogs.akamai.com/2017/05/spotlight-on-malware-dga-communication-technique.html, acesso em 2017-12-20. 85

Apêndice A

Arquivo de configuração principal para o Flume

```
#Configuracoes padrao FLUME
#Sources, channels, and sinks are difined per agent name

tier1.sources = source1
tier1.channels = channel1 channel2
tier1.sinks = sink1 sink2

#Configuracao dos interceptors
tier1.sources.source1.interceptors = i1 i2
tier1.sources.source1.interceptors.i1.type = host
tier1.sources.source1.interceptors.i1.hostHeader = host
tier1.sources.source1.interceptors.i2.type = timestamp

#Configuracao dos sources
tier1.sources.source1.type = http
tier1.sources.source1.bind = 0.0.0.0
tier1.sources.source1.port = 5140
# JSONHandler is the default for the httpsource #
tier1.sources.source1.handler = org.apache.flume.source.http.JSONHandler
tier1.sources.source1.channels = channel1 channel2
tier1.sources.source1.selector.type = replicating

#Configuracao do channel
# Kafka Channel Configuration
tier1.channels.channel1.type                 = org.apache.flume.channel.
    kafka.KafkaChannel
tier1.channels.channel1.capacity             = 10000
tier1.channels.channel1.transactionCapacity = 1000
tier1.channels.channel1.brokerList           = ip_kafka_broker:9092,
```

```
29 tier1.channels.channel1.kafka.topic                    = flume.auths
30 tier1.channels.channel1.zookeeperConnect     = ip_zookeeper:2181
31 tier1.channels.channel1.groupId = flume1
32
33 tier1.channels.channel2.type = file
34 tier1.channels.channel2.transactionCapacity = 10000
35 tier1.channels.channel2.overflowCapacity = 1000000
36 tier1.channels.channel2.checkpointDir = /home/flume/checkpoint
37 tier1.channels.channel2.dataDirs = /home/flume/data
38
39 #Configuracao da sink1
40 tier1.sinks.sink1.type = org.apache.flume.sink.hbase.AsyncHBaseSink
41 tier1.sinks.sink1.table         = norseCorp
42 tier1.sinks.sink1.columnFamily = attacks
43 tier1.sinks.sink1.serializer = org.apache.flume.sink.hbase.
       SplittingSerializer
44 tier1.sinks.sink1.serializer.columns=time,attacker_ip,attacker_city,
       attacker_contry_code,attacker_latitude,attacker_longitude,attacker_org,
       target_port,target_city,target_contry_code,target_latitude,
       target_longitude
45 tier1.sinks.sink1.channel = channel2
46
47 #Configuracao da sink2
48 tier1.sinks.sink2.type          = org.apache.flume.sink.solr.morphline.
       MorphlineSolrSink
49 tier1.sinks.sink2.channel       = channel1
50 tier1.sinks.sink2.morphlineFile = morphlines.conf
```

Apêndice B

Arquivo de configuração para a integração do SOLR com o FLUME

```
# Specify server locations in a SOLR_LOCATOR variable;
# used later in variable substitutions
# Change the zkHost to point to your own Zookeeper quorum
SOLR_LOCATOR : {
    # Name of solr collection
    collection : norseCorpCollection
    # ZooKeeper ensemble
    zkHost :  "$ZK_HOST"
}

# Specify an array of one or more morphlines, each of which defines an ETL
# transformation chain. A morphline consists of one or more (potentially
# nested) commands. A morphline is a way to consume records (e.g. Flume events,
# HDFS files or blocks), turn them into a stream of records, and pipe the stream
# of records through a set of easily configurable transformations on it's way to
# Solr (or a MapReduceIndexerTool RecordWriter that feeds via a Reducer into Solr).
morphlines : [
{
    # Name used to identify a morphline. E.g. used if there are multiple morphlines in a
    # morphline config file
    id : morphline1
    # Import all morphline commands in these java packages and their subpackages.
```

```
23    # Other commands that may be present on the classpath are not visible
      to this morphline.
24    importCommands : ["org.kitesdk.**", "org.apache.solr.**"]
25    commands : [
26    {
27        ## Read the email stream and break it up into individual messages.
28        ## The beginning of a message is marked by regex clause below
29        ## The reason we use this command is that one event can have
      multiple
30        ## messages
31        readCSV {
32          separator:   ";"
33             columns:   [time,attacker_ip,attacker_city,attacker_contry_code,
      attacker_latitude,attacker_longitude,attacker_org,target_port,
      target_city,target_contry_code,target_latitude,target_longitude]
34             ignoreFirstLine : false
35             quoteChar : ""
36             commentPrefix : ""
37             trim : true
38             charset : UTF-8
39        }
40    }
41    #{logInfo { format : "BODY : {}", args : ["@{}"] } }
42    # add Unique ID, in case our message_id field from above is not present
43    {
44        generateUUID {
45            field:id
46        }
47    }
48
49    # Consume the output record of the previous command and pipe another
50    # record downstream.
51    #
52    # This command sanitizes record fields that are unknown to Solr schema.
      xml
53    # by deleting them. Recall that Solr throws an exception on any attempt
       to
54    # load a document that contains a field that isn't specified in schema.
      xml
55    {
56        sanitizeUnknownSolrFields {
57            # Location from which to fetch Solr schema
58            solrLocator : ${SOLR_LOCATOR}
59        }
60    }
```

```
61
62      # load the record into a SolrServer or MapReduce SolrOutputFormat.
63      {
64          loadSolr {
65              solrLocator : ${SOLR_LOCATOR}
66          }
67      }
68      ]
69  }
70  ]
```

Apêndice C

Arquivo de configuração SOLR *Schema* para definição dos tipos de dados

```
<?xml version="1.0" encoding="UTF-8" ?>

<schema name="example" version="1.5">
  <!-- attribute "name" is the name of this schema and is only used for
    display purposes.
       version="x.y" is Solr's version number for the schema syntax and
       semantics.  It should not normally be changed by applications.

       1.0: multiValued attribute did not exist, all fields are multiValued
            by nature
       1.1: multiValued attribute introduced, false by default
       1.2: omitTermFreqAndPositions attribute introduced, true by default
            except for text fields.
       1.3: removed optional field compress feature
       1.4: autoGeneratePhraseQueries attribute introduced to drive
    QueryParser
            behavior when a single string produces multiple tokens.
    Defaults
            to off for version >= 1.4
       1.5: omitNorms defaults to true for primitive field types
            (int, float, boolean, string...)
     -->

 <fields>
   <!-- Valid attributes for fields:
     name: mandatory - the name for the field
```

```
24      type: mandatory - the name of a field type from the
25        <types> fieldType section
26      indexed: true if this field should be indexed (searchable or sortable)
27      stored: true if this field should be retrievable
28      docValues: true if this field should have doc values. Doc values are
29        useful for faceting, grouping, sorting and function queries.
    Although not
30        required, doc values will make the index faster to load, more
31        NRT-friendly and more memory-efficient. They however come with some
32        limitations: they are currently only supported by StrField,
    UUIDField
33        and all Trie*Fields, and depending on the field type, they might
34        require the field to be single-valued, be required or have a default
35        value (check the documentation of the field type you're interested
    in
36        for more information)
37      multiValued: true if this field may contain multiple values per
    document
38      omitNorms: (expert) set to true to omit the norms associated with
39        this field (this disables length normalization and index-time
40        boosting for the field, and saves some memory). Only full-text
41        fields or fields that need an index-time boost need norms.
42        Norms are omitted for primitive (non-analyzed) types by default.
43      termVectors: [false] set to true to store the term vector for a
44        given field.
45        When using MoreLikeThis, fields used for similarity should be
46        stored for best performance.
47      termPositions: Store position information with the term vector.
48        This will increase storage costs.
49      termOffsets: Store offset information with the term vector. This
50        will increase storage costs.
51      required: The field is required. It will throw an error if the
52        value does not exist
53      default: a value that should be used if no value is specified
54        when adding a document.
55    -->
56
57    <!-- field names should consist of alphanumeric or underscore characters
       only and
58      not start with a digit. This is not currently strictly enforced,
59      but other field names will not have first class support from all
    components
60      and back compatibility is not guaranteed. Names with both leading
    and
61      trailing underscores (e.g. _version_) are reserved.
```

```
62      -->
63
64      <field name="id" type="string" indexed="true" stored="true" required="
        true" multiValued="false" />
65      <field name="time" type="tdate" indexed="true" stored="true" />
66      <field name="record" type="text_general" indexed="true" stored="false"
        multiValued="true"/>
67
68      <!-- Informacoes do atacante -->
69      <field name="attacker_ip" type="string" indexed="true" stored="true" />
70      <field name="attacker_city" type="string" indexed="true" stored="true" /
        >
71      <field name="attacker_contry_code" type="string" indexed="true" stored="
        true" />
72      <field name="attacker_latitude" type="float" indexed="true" stored="true
        " />
73      <field name="attacker_longitude" type="float" indexed="true" stored="
        true" />
74      <field name="attacker_org" type="string" indexed="true" stored="true" />
75      <field name="target_port" type="int" indexed="true" stored="true" />
76
77      <!-- Informacoes do alvo -->
78      <field name="target_city" type="string" indexed="true" stored="true" />
79      <field name="target_contry_code" type="string" indexed="true" stored="
        true" />
80      <field name="target_latitude" type="float" indexed="true" stored="true"
        />
81      <field name="target_longitude" type="float" indexed="true" stored="true"
         />
82
83      <field name="_version_" type="long" indexed="true" stored="true"/>
84      <dynamicField name="ignored_*" type="ignored"/>
85
86  </fields>
87
88
89  <!-- Field to use to determine and enforce document uniqueness.
90       Unless this field is marked with required="false", it will be a
        required field
91      -->
92  <uniqueKey>id</uniqueKey>
93
94
95   <copyField source="time" dest="record"/>
96   <copyField source="attacker_ip" dest="record"/>
```

```
97   <copyField source="attacker_city" dest="record"/>
98   <copyField source="attacker_contry_code" dest="record"/>
99   <copyField source="attacker_latitude" dest="record"/>
100  <copyField source="attacker_longitude" dest="record"/>
101  <copyField source="attacker_org" dest="record"/>
102  <copyField source="target_port" dest="record"/>
103  <copyField source="target_city" dest="record"/>
104  <copyField source="target_contry_code" dest="record"/>
105  <copyField source="target_latitude" dest="record"/>
106  <copyField source="target_longitude" dest="record"/>
107
108 <!-- DEPRECATED: The defaultSearchField is consulted by various query
       parsers when
109   parsing a query string that isn't explicit about the field.  Machine (non
       -user)
110   generated queries are best made explicit, or they can use the "df"
       request parameter
111   which takes precedence over this.
112   Note: Un-commenting defaultSearchField will be insufficient if your
       request handler
113   in solrconfig.xml defines "df", which takes precedence. That would need
       to be removed.
114 <defaultSearchField>text</defaultSearchField> -->
115
116 <!-- DEPRECATED: The defaultOperator (AND|OR) is consulted by various
       query parsers
117   when parsing a query string to determine if a clause of the query should
       be marked as
118   required or optional, assuming the clause isn't already marked by some
       operator.
119   The default is OR, which is generally assumed so it is not a good idea to
       change it
120   globally here.  The "q.op" request parameter takes precedence over this.
121 <solrQueryParser defaultOperator="OR"/> -->
122
123  <!-- copyField commands copy one field to another at the time a document
124         is added to the index.  It's used either to index the same field
       differently,
125         or to add multiple fields to the same field for easier/faster
       searching.  -->
126
127
128  <types>
129    <!-- field type definitions. The "name" attribute is
130       just a label to be used by field definitions.  The "class"
```

```
131         attribute and any other attributes determine the real
132         behavior of the fieldType.
133           Class names starting with "solr" refer to java classes in a
134         standard package such as org.apache.solr.analysis
135       -->
136
137       <!-- The StrField type is not analyzed, but indexed/stored verbatim.
138         It supports doc values but in that case the field needs to be
139         single-valued and either required or have a default value.
140       -->
141       <fieldType name="string" class="solr.StrField" sortMissingLast="true"
          />
142
143       <!-- boolean type: "true" or "false" -->
144       <fieldType name="boolean" class="solr.BoolField" sortMissingLast="true
          "/>
145
146       <!-- sortMissingLast and sortMissingFirst attributes are optional
          attributes are
147            currently supported on types that are sorted internally as strings
148            and on numeric types.
149         This includes "string","boolean", and, as of 3.5 (and 4.x),
150         int, float, long, date, double, including the "Trie" variants.
151         - If sortMissingLast="true", then a sort on this field will cause
          documents
152            without the field to come after documents with the field,
153            regardless of the requested sort order (asc or desc).
154         - If sortMissingFirst="true", then a sort on this field will cause
          documents
155            without the field to come before documents with the field,
156            regardless of the requested sort order.
157         - If sortMissingLast="false" and sortMissingFirst="false" (the
          default),
158            then default lucene sorting will be used which places docs without
           the
159            field first in an ascending sort and last in a descending sort.
160       -->
161
162       <!--
163         Default numeric field types. For faster range queries, consider the
          tint/tfloat/tlong/tdouble types.
164
165         These fields support doc values, but they require the field to be
166         single-valued and either be required or have a default value.
167       -->
```

```
168     <fieldType name="int" class="solr.TrieIntField" precisionStep="0"
        positionIncrementGap="0"/>
169     <fieldType name="float" class="solr.TrieFloatField" precisionStep="0"
        positionIncrementGap="0"/>
170     <fieldType name="long" class="solr.TrieLongField" precisionStep="0"
        positionIncrementGap="0"/>
171     <fieldType name="double" class="solr.TrieDoubleField" precisionStep="0"
         positionIncrementGap="0"/>
172
173     <!--
174      Numeric field types that index each value at various levels of
        precision
175      to accelerate range queries when the number of values between the
        range
176      endpoints is large. See the javadoc for NumericRangeQuery for internal
177      implementation details.
178
179      Smaller precisionStep values (specified in bits) will lead to more
        tokens
180      indexed per value, slightly larger index size, and faster range
        queries.
181      A precisionStep of 0 disables indexing at different precision levels.
182     -->
183     <fieldType name="tint" class="solr.TrieIntField" precisionStep="8"
        positionIncrementGap="0"/>
184     <fieldType name="tfloat" class="solr.TrieFloatField" precisionStep="8"
        positionIncrementGap="0"/>
185     <fieldType name="tlong" class="solr.TrieLongField" precisionStep="8"
        positionIncrementGap="0"/>
186     <fieldType name="tdouble" class="solr.TrieDoubleField" precisionStep
        ="8" positionIncrementGap="0"/>
187
188     <!-- The format for this date field is of the form 1995-12-31T23:59:59Z
        , and
189          is a more restricted form of the canonical representation of
        dateTime
190          http://www.w3.org/TR/xmlschema-2/#dateTime
191          The trailing "Z" designates UTC time and is mandatory.
192          Optional fractional seconds are allowed: 1995-12-31T23:59:59.999Z
193          All other components are mandatory.
194
195          Expressions can also be used to denote calculations that should be
196          performed relative to "NOW" to determine the value, ie...
197
198               NOW/HOUR
```

```
199                 ... Round to the start of the current hour
200             NOW-1DAY
201                 ... Exactly 1 day prior to now
202             NOW/DAY+6MONTHS+3DAYS
203                 ... 6 months and 3 days in the future from the start of
204                     the current day
205
206         Consult the DateField javadocs for more information.
207
208         Note: For faster range queries, consider the tdate type
209       -->
210     <fieldType name="date" class="solr.TrieDateField" precisionStep="0"
    positionIncrementGap="0"/>
211
212     <!-- A Trie based date field for faster date range queries and date
    faceting. -->
213     <fieldType name="tdate" class="solr.TrieDateField" precisionStep="6"
    positionIncrementGap="0"/>
214
215
216     <!--Binary data type. The data should be sent/retrieved in as Base64
    encoded Strings -->
217     <fieldtype name="binary" class="solr.BinaryField"/>
218
219     <!--
220       Note:
221       These should only be used for compatibility with existing indexes (
    created with lucene or older Solr versions).
222       Use Trie based fields instead. As of Solr 3.5 and 4.x, Trie based
    fields support sortMissingFirst/Last
223
224       Plain numeric field types that store and index the text
225       value verbatim (and hence don't correctly support range queries,
    since the
226       lexicographic ordering isn't equal to the numeric ordering)
227     -->
228     <fieldType name="pint" class="solr.IntField"/>
229     <fieldType name="plong" class="solr.LongField"/>
230     <fieldType name="pfloat" class="solr.FloatField"/>
231     <fieldType name="pdouble" class="solr.DoubleField"/>
232     <fieldType name="pdate" class="solr.DateField" sortMissingLast="true"/>
233
234     <!-- The "RandomSortField" is not used to store or search any
235          data.  You can declare fields of this type it in your schema
236          to generate pseudo-random orderings of your docs for sorting
```

```
237         or function purposes.  The ordering is generated based on the
    field
238         name and the version of the index. As long as the index version
239         remains unchanged, and the same field name is reused,
240         the ordering of the docs will be consistent.
241         If you want different psuedo-random orderings of documents,
242         for the same version of the index, use a dynamicField and
243         change the field name in the request.
244      -->
245    <fieldType name="random" class="solr.RandomSortField" indexed="true" />
246
247    <!-- solr.TextField allows the specification of custom text analyzers
248         specified as a tokenizer and a list of token filters. Different
249         analyzers may be specified for indexing and querying.
250
251         The optional positionIncrementGap puts space between multiple
    fields of
252         this type on the same document, with the purpose of preventing
    false phrase
253         matching across fields.
254
255         For more info on customizing your analyzer chain, please see
256         http://wiki.apache.org/solr/AnalyzersTokenizersTokenFilters
257      -->
258
259    <!-- One can also specify an existing Analyzer class that has a
260         default constructor via the class attribute on the analyzer
    element.
261         Example:
262    <fieldType name="text_greek" class="solr.TextField">
263      <analyzer class="org.apache.lucene.analysis.el.GreekAnalyzer"/>
264    </fieldType>
265    -->
266
267    <!-- A text field that only splits on whitespace for exact matching of
    words -->
268    <fieldType name="text_ws" class="solr.TextField" positionIncrementGap
    ="100">
269      <analyzer>
270        <tokenizer class="solr.WhitespaceTokenizerFactory"/>
271      </analyzer>
272    </fieldType>
273
274    <!-- A general text field that has reasonable, generic
275         cross-language defaults: it tokenizes with StandardTokenizer,
```

```
276     removes stop words from case-insensitive "stopwords.txt"
277     (empty by default), and down cases.  At query time only, it
278     also applies synonyms. -->
279     <fieldType name="text_general" class="solr.TextField"
        positionIncrementGap="100">
280       <analyzer type="index">
281         <tokenizer class="solr.StandardTokenizerFactory"/>
282         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
        stopwords.txt" enablePositionIncrements="true" />
283         <!-- in this example, we will only use synonyms at query time
284         <filter class="solr.SynonymFilterFactory" synonyms="index_synonyms.
        txt" ignoreCase="true" expand="false"/>
285         -->
286         <filter class="solr.LowerCaseFilterFactory"/>
287       </analyzer>
288       <analyzer type="query">
289         <tokenizer class="solr.StandardTokenizerFactory"/>
290         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
        stopwords.txt" enablePositionIncrements="true" />
291         <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
        ignoreCase="true" expand="true"/>
292         <filter class="solr.LowerCaseFilterFactory"/>
293       </analyzer>
294     </fieldType>
295
296     <!-- A text field with defaults appropriate for English: it
297          tokenizes with StandardTokenizer, removes English stop words
298          (lang/stopwords_en.txt), down cases, protects words from protwords
        .txt, and
299          finally applies Porter's stemming.  The query time analyzer
300          also applies synonyms from synonyms.txt. -->
301     <fieldType name="text_en" class="solr.TextField" positionIncrementGap="
        100">
302       <analyzer type="index">
303         <tokenizer class="solr.StandardTokenizerFactory"/>
304         <!-- in this example, we will only use synonyms at query time
305         <filter class="solr.SynonymFilterFactory" synonyms="index_synonyms.
        txt" ignoreCase="true" expand="false"/>
306         -->
307         <!-- Case insensitive stop word removal.
308           add enablePositionIncrements=true in both the index and query
309           analyzers to leave a 'gap' for more accurate phrase queries.
310         -->
311         <filter class="solr.StopFilterFactory"
312                 ignoreCase="true"
```

```
313                 words="lang/stopwords_en.txt"
314                 enablePositionIncrements="true"
315                 />
316         <filter class="solr.LowerCaseFilterFactory"/>
317   <filter class="solr.EnglishPossessiveFilterFactory"/>
318         <filter class="solr.KeywordMarkerFilterFactory" protected="
      protwords.txt"/>
319   <!-- Optionally you may want to use this less aggressive stemmer instead
      of PorterStemFilterFactory:
320         <filter class="solr.EnglishMinimalStemFilterFactory"/>
321   -->
322         <filter class="solr.PorterStemFilterFactory"/>
323       </analyzer>
324       <analyzer type="query">
325         <tokenizer class="solr.StandardTokenizerFactory"/>
326         <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
      ignoreCase="true" expand="true"/>
327         <filter class="solr.StopFilterFactory"
328                 ignoreCase="true"
329                 words="lang/stopwords_en.txt"
330                 enablePositionIncrements="true"
331                 />
332         <filter class="solr.LowerCaseFilterFactory"/>
333   <filter class="solr.EnglishPossessiveFilterFactory"/>
334         <filter class="solr.KeywordMarkerFilterFactory" protected="
      protwords.txt"/>
335   <!-- Optionally you may want to use this less aggressive stemmer instead
      of PorterStemFilterFactory:
336         <filter class="solr.EnglishMinimalStemFilterFactory"/>
337   -->
338         <filter class="solr.PorterStemFilterFactory"/>
339       </analyzer>
340     </fieldType>
341
342     <!-- A text field with defaults appropriate for English, plus
343    aggressive word-splitting and autophrase features enabled.
344    This field is just like text_en, except it adds
345    WordDelimiterFilter to enable splitting and matching of
346    words on case-change, alpha numeric boundaries, and
347    non-alphanumeric chars.  This means certain compound word
348    cases will work, for example query "wi fi" will match
349    document "WiFi" or "wi-fi".
350         -->
351     <fieldType name="text_en_splitting" class="solr.TextField"
      positionIncrementGap="100" autoGeneratePhraseQueries="true">
```

```
352       <analyzer type="index">
353         <tokenizer class="solr.WhitespaceTokenizerFactory"/>
354         <!-- in this example, we will only use synonyms at query time
355         <filter class="solr.SynonymFilterFactory" synonyms="index_synonyms.
    txt" ignoreCase="true" expand="false"/>
356         -->
357         <!-- Case insensitive stop word removal.
358           add enablePositionIncrements=true in both the index and query
359           analyzers to leave a 'gap' for more accurate phrase queries.
360         -->
361         <filter class="solr.StopFilterFactory"
362                 ignoreCase="true"
363                 words="lang/stopwords_en.txt"
364                 enablePositionIncrements="true"
365                 />
366         <filter class="solr.WordDelimiterFilterFactory" generateWordParts="
    1" generateNumberParts="1" catenateWords="1" catenateNumbers="1"
    catenateAll="0" splitOnCaseChange="1"/>
367         <filter class="solr.LowerCaseFilterFactory"/>
368         <filter class="solr.KeywordMarkerFilterFactory" protected="
    protwords.txt"/>
369         <filter class="solr.PorterStemFilterFactory"/>
370       </analyzer>
371       <analyzer type="query">
372         <tokenizer class="solr.WhitespaceTokenizerFactory"/>
373         <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
    ignoreCase="true" expand="true"/>
374         <filter class="solr.StopFilterFactory"
375                 ignoreCase="true"
376                 words="lang/stopwords_en.txt"
377                 enablePositionIncrements="true"
378                 />
379         <filter class="solr.WordDelimiterFilterFactory" generateWordParts="
    1" generateNumberParts="1" catenateWords="0" catenateNumbers="0"
    catenateAll="0" splitOnCaseChange="1"/>
380         <filter class="solr.LowerCaseFilterFactory"/>
381         <filter class="solr.KeywordMarkerFilterFactory" protected="
    protwords.txt"/>
382         <filter class="solr.PorterStemFilterFactory"/>
383       </analyzer>
384     </fieldType>
385
386     <!-- Less flexible matching, but less false matches.  Probably not
    ideal for product names,
```

```
387         but may be good for SKUs.  Can insert dashes in the wrong place
    and still match. -->
388     <fieldType name="text_en_splitting_tight" class="solr.TextField"
    positionIncrementGap="100" autoGeneratePhraseQueries="true">
389       <analyzer>
390         <tokenizer class="solr.WhitespaceTokenizerFactory"/>
391         <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
    ignoreCase="true" expand="false"/>
392         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
    lang/stopwords_en.txt"/>
393         <filter class="solr.WordDelimiterFilterFactory" generateWordParts="
    0" generateNumberParts="0" catenateWords="1" catenateNumbers="1"
    catenateAll="0"/>
394         <filter class="solr.LowerCaseFilterFactory"/>
395         <filter class="solr.KeywordMarkerFilterFactory" protected="
    protwords.txt"/>
396         <filter class="solr.EnglishMinimalStemFilterFactory"/>
397         <!-- this filter can remove any duplicate tokens that appear at the
     same position - sometimes
398              possible with WordDelimiterFilter in conjuncton with stemming.
     -->
399         <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
400       </analyzer>
401     </fieldType>
402
403     <!-- Just like text_general except it reverses the characters of
404    each token, to enable more efficient leading wildcard queries. -->
405     <fieldType name="text_general_rev" class="solr.TextField"
    positionIncrementGap="100">
406       <analyzer type="index">
407         <tokenizer class="solr.StandardTokenizerFactory"/>
408         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
    stopwords.txt" enablePositionIncrements="true" />
409         <filter class="solr.LowerCaseFilterFactory"/>
410         <filter class="solr.ReversedWildcardFilterFactory" withOriginal="
    true"
411            maxPosAsterisk="3" maxPosQuestion="2" maxFractionAsterisk="0.33"
    />
412       </analyzer>
413       <analyzer type="query">
414         <tokenizer class="solr.StandardTokenizerFactory"/>
415         <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
    ignoreCase="true" expand="true"/>
416         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
    stopwords.txt" enablePositionIncrements="true" />
```

```
417         <filter class="solr.LowerCaseFilterFactory"/>
418       </analyzer>
419     </fieldType>
420
421     <!-- charFilter + WhitespaceTokenizer  -->
422     <!--
423     <fieldType name="text_char_norm" class="solr.TextField"
    positionIncrementGap="100" >
424       <analyzer>
425         <charFilter class="solr.MappingCharFilterFactory" mapping="mapping-
    ISOLatin1Accent.txt"/>
426         <tokenizer class="solr.WhitespaceTokenizerFactory"/>
427       </analyzer>
428     </fieldType>
429     -->
430
431     <!-- This is an example of using the KeywordTokenizer along
432          With various TokenFilterFactories to produce a sortable field
433          that does not include some properties of the source text
434       -->
435     <fieldType name="alphaOnlySort" class="solr.TextField" sortMissingLast=
    "true" omitNorms="true">
436       <analyzer>
437         <!-- KeywordTokenizer does no actual tokenizing, so the entire
438              input string is preserved as a single token
439           -->
440         <tokenizer class="solr.KeywordTokenizerFactory"/>
441         <!-- The LowerCase TokenFilter does what you expect, which can be
442              when you want your sorting to be case insensitive
443           -->
444         <filter class="solr.LowerCaseFilterFactory" />
445         <!-- The TrimFilter removes any leading or trailing whitespace -->
446         <filter class="solr.TrimFilterFactory" />
447         <!-- The PatternReplaceFilter gives you the flexibility to use
448              Java Regular expression to replace any sequence of characters
449              matching a pattern with an arbitrary replacement string,
450              which may include back references to portions of the original
451              string matched by the pattern.
452
453              See the Java Regular Expression documentation for more
454              information on pattern and replacement string syntax.
455
456              http://java.sun.com/j2se/1.6.0/docs/api/java/util/regex/
    package-summary.html
457         -->
```

```
458         <filter class="solr.PatternReplaceFilterFactory"
459                 pattern="([^a-z])" replacement="" replace="all"
460         />
461       </analyzer>
462     </fieldType>
463
464     <fieldtype name="phonetic" stored="false" indexed="true" class="solr.
    TextField" >
465       <analyzer>
466         <tokenizer class="solr.StandardTokenizerFactory"/>
467         <filter class="solr.DoubleMetaphoneFilterFactory" inject="false"/>
468       </analyzer>
469     </fieldtype>
470
471     <fieldtype name="payloads" stored="false" indexed="true" class="solr.
    TextField" >
472       <analyzer>
473         <tokenizer class="solr.WhitespaceTokenizerFactory"/>
474         <!--
475          The DelimitedPayloadTokenFilter can put payloads on tokens... for
    example,
476          a token of "foo|1.4"  would be indexed as "foo" with a payload of
    1.4f
477          Attributes of the DelimitedPayloadTokenFilterFactory :
478           "delimiter" - a one character delimiter. Default is | (pipe)
479     "encoder" - how to encode the following value into a playload
480        float -> org.apache.lucene.analysis.payloads.FloatEncoder,
481        integer -> o.a.l.a.p.IntegerEncoder
482        identity -> o.a.l.a.p.IdentityEncoder
483              Fully Qualified class name implementing PayloadEncoder, Encoder
     must have a no arg constructor.
484           -->
485         <filter class="solr.DelimitedPayloadTokenFilterFactory" encoder="
    float"/>
486       </analyzer>
487     </fieldtype>
488
489     <!-- lowercases the entire field value, keeping it as a single token.
    -->
490     <fieldType name="lowercase" class="solr.TextField" positionIncrementGap
    ="100">
491       <analyzer>
492         <tokenizer class="solr.KeywordTokenizerFactory"/>
493         <filter class="solr.LowerCaseFilterFactory" />
494       </analyzer>
```

```
    </fieldType>

    <!--
       Example of using PathHierarchyTokenizerFactory at index time, so
       queries for paths match documents at that path, or in descendent
     paths
    -->
    <fieldType name="descendent_path" class="solr.TextField">
      <analyzer type="index">
  <tokenizer class="solr.PathHierarchyTokenizerFactory" delimiter="/" />
      </analyzer>
      <analyzer type="query">
  <tokenizer class="solr.KeywordTokenizerFactory" />
      </analyzer>
    </fieldType>
    <!--
       Example of using PathHierarchyTokenizerFactory at query time, so
       queries for paths match documents at that path, or in ancestor paths
    -->
    <fieldType name="ancestor_path" class="solr.TextField">
      <analyzer type="index">
  <tokenizer class="solr.KeywordTokenizerFactory" />
      </analyzer>
      <analyzer type="query">
  <tokenizer class="solr.PathHierarchyTokenizerFactory" delimiter="/" />
      </analyzer>
    </fieldType>

    <!-- since fields of this type are by default not stored or indexed,
         any data added to them will be ignored outright.  -->
    <fieldtype name="ignored" stored="false" indexed="false" multiValued="
    true" class="solr.StrField" />

    <!-- This point type indexes the coordinates as separate fields (
    subFields)
       If subFieldType is defined, it references a type, and a dynamic field
       definition is created matching *___<typename>.  Alternately, if
       subFieldSuffix is defined, that is used to create the subFields.
       Example: if subFieldType="double", then the coordinates would be
         indexed in fields myloc_0___double,myloc_1___double.
       Example: if subFieldSuffix="_d" then the coordinates would be indexed
         in fields myloc_0_d,myloc_1_d
       The subFields are an implementation detail of the fieldType, and end
       users normally should not need to know about them.
     -->
```

```
537     <fieldType name="point" class="solr.PointType" dimension="2"
        subFieldSuffix="_d"/>
538
539     <!-- A specialized field for geospatial search. If indexed, this
        fieldType must not be multivalued. -->
540     <fieldType name="location" class="solr.LatLonType" subFieldSuffix="
        _coordinate"/>
541
542     <!-- An alternative geospatial field type new to Solr 4.  It supports
        multiValued and polygon shapes.
543       For more information about this and other Spatial fields new to Solr
        4, see:
544       http://wiki.apache.org/solr/SolrAdaptersForLuceneSpatial4
545     -->
546     <fieldType name="location_rpt" class="solr.
        SpatialRecursivePrefixTreeFieldType"
547         geo="true" distErrPct="0.025" maxDistErr="0.000009" units="degrees"
         />
548
549    <!-- Money/currency field type. See http://wiki.apache.org/solr/
        MoneyFieldType
550         Parameters:
551           defaultCurrency: Specifies the default currency if none specified
        . Defaults to "USD"
552           precisionStep:   Specifies the precisionStep for the TrieLong
        field used for the amount
553           providerClass:   Lets you plug in other exchange provider
        backend:
554                            solr.FileExchangeRateProvider is the default and
         takes one parameter:
555                              currencyConfig: name of an xml file holding
        exchange rates
556                            solr.OpenExchangeRatesOrgProvider uses rates
        from openexchangerates.org:
557                              ratesFileLocation: URL or path to rates JSON
        file (default latest.json on the web)
558                              refreshInterval: Number of minutes between
        each rates fetch (default: 1440, min: 60)
559    -->
560     <fieldType name="currency" class="solr.CurrencyField" precisionStep="8"
         defaultCurrency="USD" currencyConfig="currency.xml" />
561
562
563
```

```
564    <!-- some examples for different languages (generally ordered by ISO
       code) -->
565
566     <!-- Arabic -->
567     <fieldType name="text_ar" class="solr.TextField" positionIncrementGap="
       100">
568       <analyzer>
569         <tokenizer class="solr.StandardTokenizerFactory"/>
570         <!-- for any non-arabic -->
571         <filter class="solr.LowerCaseFilterFactory"/>
572         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
       lang/stopwords_ar.txt" enablePositionIncrements="true"/>
573         <!-- normalizes to etc -->
574         <filter class="solr.ArabicNormalizationFilterFactory"/>
575         <filter class="solr.ArabicStemFilterFactory"/>
576       </analyzer>
577     </fieldType>
578
579     <!-- Bulgarian -->
580     <fieldType name="text_bg" class="solr.TextField" positionIncrementGap="
       100">
581       <analyzer>
582         <tokenizer class="solr.StandardTokenizerFactory"/>
583         <filter class="solr.LowerCaseFilterFactory"/>
584         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
       lang/stopwords_bg.txt" enablePositionIncrements="true"/>
585         <filter class="solr.BulgarianStemFilterFactory"/>
586       </analyzer>
587     </fieldType>
588
589     <!-- Catalan -->
590     <fieldType name="text_ca" class="solr.TextField" positionIncrementGap="
       100">
591       <analyzer>
592         <tokenizer class="solr.StandardTokenizerFactory"/>
593         <!-- removes l', etc -->
594         <filter class="solr.ElisionFilterFactory" ignoreCase="true"
       articles="lang/contractions_ca.txt"/>
595         <filter class="solr.LowerCaseFilterFactory"/>
596         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
       lang/stopwords_ca.txt" enablePositionIncrements="true"/>
597         <filter class="solr.SnowballPorterFilterFactory" language="Catalan
       "/>
598       </analyzer>
599     </fieldType>
```

```
600
601     <!-- CJK bigram (see text_ja for a Japanese configuration using
        morphological analysis) -->
602     <fieldType name="text_cjk" class="solr.TextField" positionIncrementGap
        ="100">
603       <analyzer>
604         <tokenizer class="solr.StandardTokenizerFactory"/>
605         <!-- normalize width before bigram, as e.g. half-width dakuten
        combine  -->
606         <filter class="solr.CJKWidthFilterFactory"/>
607         <!-- for any non-CJK -->
608         <filter class="solr.LowerCaseFilterFactory"/>
609         <filter class="solr.CJKBigramFilterFactory"/>
610       </analyzer>
611     </fieldType>
612
613     <!-- Czech -->
614     <fieldType name="text_cz" class="solr.TextField" positionIncrementGap
        ="100">
615       <analyzer>
616         <tokenizer class="solr.StandardTokenizerFactory"/>
617         <filter class="solr.LowerCaseFilterFactory"/>
618         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
        lang/stopwords_cz.txt" enablePositionIncrements="true"/>
619         <filter class="solr.CzechStemFilterFactory"/>
620       </analyzer>
621     </fieldType>
622
623     <!-- Danish -->
624     <fieldType name="text_da" class="solr.TextField" positionIncrementGap
        ="100">
625       <analyzer>
626         <tokenizer class="solr.StandardTokenizerFactory"/>
627         <filter class="solr.LowerCaseFilterFactory"/>
628         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
        lang/stopwords_da.txt" format="snowball" enablePositionIncrements="true
        "/>
629         <filter class="solr.SnowballPorterFilterFactory" language="Danish
        "/>
630       </analyzer>
631     </fieldType>
632
633     <!-- German -->
634     <fieldType name="text_de" class="solr.TextField" positionIncrementGap
        ="100">
```

```
635       <analyzer>
636         <tokenizer class="solr.StandardTokenizerFactory"/>
637         <filter class="solr.LowerCaseFilterFactory"/>
638         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
      lang/stopwords_de.txt" format="snowball" enablePositionIncrements="true
      "/>
639         <filter class="solr.GermanNormalizationFilterFactory"/>
640         <filter class="solr.GermanLightStemFilterFactory"/>
641         <!-- less aggressive: <filter class="solr.
      GermanMinimalStemFilterFactory"/> -->
642         <!-- more aggressive: <filter class="solr.
      SnowballPorterFilterFactory" language="German2"/> -->
643       </analyzer>
644     </fieldType>
645
646     <!-- Greek -->
647     <fieldType name="text_el" class="solr.TextField" positionIncrementGap
      ="100">
648       <analyzer>
649         <tokenizer class="solr.StandardTokenizerFactory"/>
650         <!-- greek specific lowercase for sigma -->
651         <filter class="solr.GreekLowerCaseFilterFactory"/>
652         <filter class="solr.StopFilterFactory" ignoreCase="false" words="
      lang/stopwords_el.txt" enablePositionIncrements="true"/>
653         <filter class="solr.GreekStemFilterFactory"/>
654       </analyzer>
655     </fieldType>
656
657     <!-- Spanish -->
658     <fieldType name="text_es" class="solr.TextField" positionIncrementGap
      ="100">
659       <analyzer>
660         <tokenizer class="solr.StandardTokenizerFactory"/>
661         <filter class="solr.LowerCaseFilterFactory"/>
662         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
      lang/stopwords_es.txt" format="snowball" enablePositionIncrements="true
      "/>
663         <filter class="solr.SpanishLightStemFilterFactory"/>
664         <!-- more aggressive: <filter class="solr.
      SnowballPorterFilterFactory" language="Spanish"/> -->
665       </analyzer>
666     </fieldType>
667
668     <!-- Basque -->
```

```
669     <fieldType name="text_eu" class="solr.TextField" positionIncrementGap
        ="100">
670       <analyzer>
671         <tokenizer class="solr.StandardTokenizerFactory"/>
672         <filter class="solr.LowerCaseFilterFactory"/>
673         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
        lang/stopwords_eu.txt" enablePositionIncrements="true"/>
674         <filter class="solr.SnowballPorterFilterFactory" language="Basque
        "/>
675       </analyzer>
676     </fieldType>
677
678     <!-- Persian -->
679     <fieldType name="text_fa" class="solr.TextField" positionIncrementGap
        ="100">
680       <analyzer>
681         <!-- for ZWNJ -->
682         <charFilter class="solr.PersianCharFilterFactory"/>
683         <tokenizer class="solr.StandardTokenizerFactory"/>
684         <filter class="solr.LowerCaseFilterFactory"/>
685         <filter class="solr.ArabicNormalizationFilterFactory"/>
686         <filter class="solr.PersianNormalizationFilterFactory"/>
687         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
        lang/stopwords_fa.txt" enablePositionIncrements="true"/>
688       </analyzer>
689     </fieldType>
690
691     <!-- Finnish -->
692     <fieldType name="text_fi" class="solr.TextField" positionIncrementGap
        ="100">
693       <analyzer>
694         <tokenizer class="solr.StandardTokenizerFactory"/>
695         <filter class="solr.LowerCaseFilterFactory"/>
696         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
        lang/stopwords_fi.txt" format="snowball" enablePositionIncrements="true
        "/>
697         <filter class="solr.SnowballPorterFilterFactory" language="Finnish
        "/>
698         <!-- less aggressive: <filter class="solr.
        FinnishLightStemFilterFactory"/> -->
699       </analyzer>
700     </fieldType>
701
702     <!-- French -->
```

```
703     <fieldType name="text_fr" class="solr.TextField" positionIncrementGap
        ="100">
704       <analyzer>
705         <tokenizer class="solr.StandardTokenizerFactory"/>
706         <!-- removes l', etc -->
707         <filter class="solr.ElisionFilterFactory" ignoreCase="true"
        articles="lang/contractions_fr.txt"/>
708         <filter class="solr.LowerCaseFilterFactory"/>
709         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
        lang/stopwords_fr.txt" format="snowball" enablePositionIncrements="true"
        />
710         <filter class="solr.FrenchLightStemFilterFactory"/>
711         <!-- less aggressive: <filter class="solr.
        FrenchMinimalStemFilterFactory"/> -->
712         <!-- more aggressive: <filter class="solr.
        SnowballPorterFilterFactory" language="French"/> -->
713       </analyzer>
714     </fieldType>
715
716     <!-- Irish -->
717     <fieldType name="text_ga" class="solr.TextField" positionIncrementGap="
        100">
718       <analyzer>
719         <tokenizer class="solr.StandardTokenizerFactory"/>
720         <!-- removes d', etc -->
721         <filter class="solr.ElisionFilterFactory" ignoreCase="true"
        articles="lang/contractions_ga.txt"/>
722         <!-- removes n-, etc. position increments is intentionally false!
        -->
723         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
        lang/hyphenations_ga.txt" enablePositionIncrements="false"/>
724         <filter class="solr.IrishLowerCaseFilterFactory"/>
725         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
        lang/stopwords_ga.txt" enablePositionIncrements="true"/>
726         <filter class="solr.SnowballPorterFilterFactory" language="Irish"/>
727       </analyzer>
728     </fieldType>
729
730     <!-- Galician -->
731     <fieldType name="text_gl" class="solr.TextField" positionIncrementGap
        ="100">
732       <analyzer>
733         <tokenizer class="solr.StandardTokenizerFactory"/>
734         <filter class="solr.LowerCaseFilterFactory"/>
```

```
735         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
    lang/stopwords_gl.txt" enablePositionIncrements="true"/>
736         <filter class="solr.GalicianStemFilterFactory"/>
737         <!-- less aggressive: <filter class="solr.
    GalicianMinimalStemFilterFactory"/> -->
738       </analyzer>
739     </fieldType>
740 
741     <!-- Hindi -->
742     <fieldType name="text_hi" class="solr.TextField" positionIncrementGap
    ="100">
743       <analyzer>
744         <tokenizer class="solr.StandardTokenizerFactory"/>
745         <filter class="solr.LowerCaseFilterFactory"/>
746         <!-- normalizes unicode representation -->
747         <filter class="solr.IndicNormalizationFilterFactory"/>
748         <!-- normalizes variation in spelling -->
749         <filter class="solr.HindiNormalizationFilterFactory"/>
750         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
    lang/stopwords_hi.txt" enablePositionIncrements="true"/>
751         <filter class="solr.HindiStemFilterFactory"/>
752       </analyzer>
753     </fieldType>
754 
755     <!-- Hungarian -->
756     <fieldType name="text_hu" class="solr.TextField" positionIncrementGap
    ="100">
757       <analyzer>
758         <tokenizer class="solr.StandardTokenizerFactory"/>
759         <filter class="solr.LowerCaseFilterFactory"/>
760         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
    lang/stopwords_hu.txt" format="snowball" enablePositionIncrements="true
    "/>
761         <filter class="solr.SnowballPorterFilterFactory" language="
    Hungarian"/>
762         <!-- less aggressive: <filter class="solr.
    HungarianLightStemFilterFactory"/> -->
763       </analyzer>
764     </fieldType>
765 
766     <!-- Armenian -->
767     <fieldType name="text_hy" class="solr.TextField" positionIncrementGap
    ="100">
768       <analyzer>
769         <tokenizer class="solr.StandardTokenizerFactory"/>
```

```
770         <filter class="solr.LowerCaseFilterFactory"/>
771         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
    lang/stopwords_hy.txt" enablePositionIncrements="true"/>
772         <filter class="solr.SnowballPorterFilterFactory" language="Armenian
    "/>
773       </analyzer>
774     </fieldType>
775
776     <!-- Indonesian -->
777     <fieldType name="text_id" class="solr.TextField" positionIncrementGap
    ="100">
778       <analyzer>
779         <tokenizer class="solr.StandardTokenizerFactory"/>
780         <filter class="solr.LowerCaseFilterFactory"/>
781         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
    lang/stopwords_id.txt" enablePositionIncrements="true"/>
782         <!-- for a less aggressive approach (only inflectional suffixes),
    set stemDerivational to false -->
783         <filter class="solr.IndonesianStemFilterFactory" stemDerivational="
    true"/>
784       </analyzer>
785     </fieldType>
786
787     <!-- Italian -->
788     <fieldType name="text_it" class="solr.TextField" positionIncrementGap
    ="100">
789       <analyzer>
790         <tokenizer class="solr.StandardTokenizerFactory"/>
791         <!-- removes l', etc -->
792         <filter class="solr.ElisionFilterFactory" ignoreCase="true"
    articles="lang/contractions_it.txt"/>
793         <filter class="solr.LowerCaseFilterFactory"/>
794         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
    lang/stopwords_it.txt" format="snowball" enablePositionIncrements="true"
    />
795         <filter class="solr.ItalianLightStemFilterFactory"/>
796         <!-- more aggressive: <filter class="solr.
    SnowballPorterFilterFactory" language="Italian"/> -->
797       </analyzer>
798     </fieldType>
799
800     <!-- Latvian -->
801     <fieldType name="text_lv" class="solr.TextField" positionIncrementGap="
    100">
802       <analyzer>
```

```
803         <tokenizer class="solr.StandardTokenizerFactory"/>
804         <filter class="solr.LowerCaseFilterFactory"/>
805         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
    lang/stopwords_lv.txt" enablePositionIncrements="true"/>
806         <filter class="solr.LatvianStemFilterFactory"/>
807       </analyzer>
808     </fieldType>
809
810     <!-- Dutch -->
811     <fieldType name="text_nl" class="solr.TextField" positionIncrementGap="
    100">
812       <analyzer>
813         <tokenizer class="solr.StandardTokenizerFactory"/>
814         <filter class="solr.LowerCaseFilterFactory"/>
815         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
    lang/stopwords_nl.txt" format="snowball" enablePositionIncrements="true"
    />
816         <filter class="solr.StemmerOverrideFilterFactory" dictionary="lang/
    stemdict_nl.txt" ignoreCase="false"/>
817         <filter class="solr.SnowballPorterFilterFactory" language="Dutch"/>
818       </analyzer>
819     </fieldType>
820
821     <!-- Norwegian -->
822     <fieldType name="text_no" class="solr.TextField" positionIncrementGap="
    100">
823       <analyzer>
824         <tokenizer class="solr.StandardTokenizerFactory"/>
825         <filter class="solr.LowerCaseFilterFactory"/>
826         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
    lang/stopwords_no.txt" format="snowball" enablePositionIncrements="true"
    />
827         <filter class="solr.SnowballPorterFilterFactory" language="
    Norwegian"/>
828         <!-- less aggressive: <filter class="solr.
    NorwegianLightStemFilterFactory"/> -->
829         <!-- singular/plural: <filter class="solr.
    NorwegianMinimalStemFilterFactory"/> -->
830       </analyzer>
831     </fieldType>
832
833     <!-- Portuguese -->
834     <fieldType name="text_pt" class="solr.TextField" positionIncrementGap="
    100">
835       <analyzer>
```

```
836         <tokenizer class="solr.StandardTokenizerFactory"/>
837         <filter class="solr.LowerCaseFilterFactory"/>
838         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
    lang/stopwords_pt.txt" format="snowball" enablePositionIncrements="true"
    />
839         <filter class="solr.PortugueseLightStemFilterFactory"/>
840         <!-- less aggressive: <filter class="solr.
    PortugueseMinimalStemFilterFactory"/> -->
841         <!-- more aggressive: <filter class="solr.
    SnowballPorterFilterFactory" language="Portuguese"/> -->
842         <!-- most aggressive: <filter class="solr.
    PortugueseStemFilterFactory"/> -->
843       </analyzer>
844     </fieldType>
845
846     <!-- Romanian -->
847     <fieldType name="text_ro" class="solr.TextField" positionIncrementGap="
    100">
848       <analyzer>
849         <tokenizer class="solr.StandardTokenizerFactory"/>
850         <filter class="solr.LowerCaseFilterFactory"/>
851         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
    lang/stopwords_ro.txt" enablePositionIncrements="true"/>
852         <filter class="solr.SnowballPorterFilterFactory" language="Romanian
    "/>
853       </analyzer>
854     </fieldType>
855
856     <!-- Russian -->
857     <fieldType name="text_ru" class="solr.TextField" positionIncrementGap="
    100">
858       <analyzer>
859         <tokenizer class="solr.StandardTokenizerFactory"/>
860         <filter class="solr.LowerCaseFilterFactory"/>
861         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
    lang/stopwords_ru.txt" format="snowball" enablePositionIncrements="true"
    />
862         <filter class="solr.SnowballPorterFilterFactory" language="Russian"
    />
863         <!-- less aggressive: <filter class="solr.
    RussianLightStemFilterFactory"/> -->
864       </analyzer>
865     </fieldType>
866
867     <!-- Swedish -->
```

```
868     <fieldType name="text_sv" class="solr.TextField" positionIncrementGap="
        100">
869       <analyzer>
870         <tokenizer class="solr.StandardTokenizerFactory"/>
871         <filter class="solr.LowerCaseFilterFactory"/>
872         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
        lang/stopwords_sv.txt" format="snowball" enablePositionIncrements="true"
        />
873         <filter class="solr.SnowballPorterFilterFactory" language="Swedish"
        />
874         <!-- less aggressive: <filter class="solr.
        SwedishLightStemFilterFactory"/> -->
875       </analyzer>
876     </fieldType>
877
878     <!-- Thai -->
879     <fieldType name="text_th" class="solr.TextField" positionIncrementGap="
        100">
880       <analyzer>
881         <tokenizer class="solr.StandardTokenizerFactory"/>
882         <filter class="solr.LowerCaseFilterFactory"/>
883         <filter class="solr.ThaiWordFilterFactory"/>
884         <filter class="solr.StopFilterFactory" ignoreCase="true" words="
        lang/stopwords_th.txt" enablePositionIncrements="true"/>
885       </analyzer>
886     </fieldType>
887
888     <!-- Turkish -->
889     <fieldType name="text_tr" class="solr.TextField" positionIncrementGap="
        100">
890       <analyzer>
891         <tokenizer class="solr.StandardTokenizerFactory"/>
892         <filter class="solr.TurkishLowerCaseFilterFactory"/>
893         <filter class="solr.StopFilterFactory" ignoreCase="false" words="
        lang/stopwords_tr.txt" enablePositionIncrements="true"/>
894         <filter class="solr.SnowballPorterFilterFactory" language="Turkish"
        />
895       </analyzer>
896     </fieldType>
897
898 </types>
899
900  <!-- Similarity is the scoring routine for each document vs. a query.
901       A custom Similarity or SimilarityFactory may be specified here, but
902       the default is fine for most applications.
```

```
903         For more info: http://wiki.apache.org/solr/SchemaXml#Similarity
904       -->
905     <!--
906       <similarity class="com.example.solr.CustomSimilarityFactory">
907         <str name="paramkey">param value</str>
908       </similarity>
909       -->
910
911 </schema>
```

Apêndice D

Arquivo de configuração para integração do Flume com HBase

```
1  package org.apache.flume.sink.hbase;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.io.File;
6  import java.io.IOException;
7  import java.io.BufferedReader;
8  import java.io.BufferedWriter;
9  import java.io.File;
10 import java.io.FileReader;
11 import java.io.FileWriter;
12
13 import org.apache.flume.Context;
14 import org.apache.flume.Event;
15 import org.apache.flume.FlumeException;
16 import org.hbase.async.AtomicIncrementRequest;
17 import org.hbase.async.PutRequest;
18 import org.apache.flume.conf.ComponentConfiguration;
19 import org.apache.flume.sink.hbase.AsyncHbaseEventSerializer;
20
21 /**
22  *  * A serializer for the AsyncHBaseSink, which splits the event body into
23  *   * multiple columns and inserts them into a row whose key is available
       in
24  *    * the headers
25  *     *
26  *      * Originally from https://blogs.apache.org/flume/entry/
       streaming_data_into_apache_hbase
27  *       * Modi/**
```

```
28  *  * A serializer for the AsyncHBaseSink, which splits the event body into
29  *  * multiple columns and inserts them into a row whose key is available
        in
30  *  * the headers
31  *
32  **/
33 public class SplittingSerializer implements AsyncHbaseEventSerializer {
34   private byte[] table;
35   private byte[] colFam;
36   private Event currentEvent;
37   private byte[][] columnNames;
38   private final List<PutRequest> puts = new ArrayList<PutRequest>();
39   private final List<AtomicIncrementRequest> incs = new ArrayList<
       AtomicIncrementRequest>();
40   private byte[] currentRowKey;
41   private final byte[] eventCountCol = "eventCount".getBytes();
42
43   @Override
44   public void initialize(byte[] table, byte[] cf) {
45     this.table = table;
46     this.colFam = cf;
47   }
48
49   @Override
50   public void setEvent(Event event) {
51
52     // Set the event and verify that the rowKey is not present
53     this.currentEvent = event;
54     String rowKeyStr = currentEvent.getHeaders().get("timestamp");
55     if (rowKeyStr == null) {
56       throw new FlumeException("No row key found in headers!");
57     }
58     currentRowKey = rowKeyStr.getBytes();
59   }
60
61   @Override
62   public List<PutRequest> getActions() {
63
64       // Split the event body and get the values for the columns
65       String eventStr = new String(currentEvent.getBody());
66       String[] cols = eventStr.split(";");
67       puts.clear();
68       for (int i = 0; i < cols.length; i++) {
69         //Generate a PutRequest for each column.
70         PutRequest req = new PutRequest(table, currentRowKey, colFam,
```

```
71            columnNames[i], cols[i].getBytes());
72            puts.add(req);
73          }
74          return puts;
75
76    }
77     @Override
78    public List<AtomicIncrementRequest> getIncrements() {
79      incs.clear();
80      //Increment the number of events received
81      incs.add(new AtomicIncrementRequest(table, "totalEvents".getBytes(),
       colFam, eventCountCol));
82      return incs;
83    }
84
85    @Override
86    public void cleanUp() {
87      table = null;
88      colFam = null;
89      currentEvent = null;
90      columnNames = null;
91      currentRowKey = null;
92    }
93
94    @Override
95    public void configure(Context context) {
96      //Get the column names from the configuration
97      String cols = new String(context.getString("columns"));
98      String[] names = cols.split(",");
99      columnNames = new byte[names.length][];
100     int i = 0;
101     for(String name : names) {
102       columnNames[i++] = name.getBytes();
103     }
104   }
105
106   @Override
107   public void configure(ComponentConfiguration conf) {
108   }
109 }
```

Apêndice E

Catcher para captura dos dados da empresa NorseCorp

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals
import websocket
from threading import Thread
import json
import datetime
import time
import os
import requests

timeout = 5
isopen = 0
threads = []
header = {
    'Content-Type': 'application/json; charset=UTF-8'
}

def formatTime(dt, precision=3):
    us = str(dt.microsecond)
    f = us[:precision] if len(us) > precision else us
    return "%d-%d-%d %d:%d:%d.%d" % (dt.year, dt.month, dt.day, dt.hour, dt
    .minute, dt.second, int(f))

def on_message(ws, message):
    global timeout,isopen,threads

    timeout -= 1
    if(timeout < 0):
        ws.close()
```

```
29      elif(message.find('"rid":1') != -1):
30          isopen = 1
31          print("Rid 1 recebido!");
32          ws.send('{"event":"#subscribe","data":{"channel":"global"},"cid":2}
        ');
33      elif(message.find('"rid":2') != -1):
34          print("Rid 2 recebido!");
35      elif(message == "#1"):
36          print("Ping recebido!");
37          ws.send("#2");
38      else:
39          timeout = 5
40          newthread = Thread_handler_capture(message)
41          newthread.start()
42          print "Tratando mensagem"
43          threads.append(newthread)
44
45  def on_error(ws, error):
46      print(error)
47
48  def on_close(ws):
49      global isopen,threads
50      isopen = 0
51      for thread in threads:
52          thread.join()
53      print("### closed ###")
54
55  def on_open(ws):
56      ws.send('{"event":"#handshake","data":{"authToken":null},"cid":1}')
57
58  def writePidFile():
59          pid = str(os.getpid())
60          currentFile = open("/var/run/websocket_norse_cloudera.pid", "w")
61          currentFile.write(pid)
62          currentFile.close()
63
64  class Thread_handler_capture(Thread):
65      def __init__(self, message):
66          Thread.__init__(self)
67          self.message = message
68      def run(self):
69          global header
70          message_json = json.loads(self.message)
71          for ataque in message_json['data']['data']:
72              aux_time = str(formatTime(datetime.datetime.utcnow()))
```

```
73              print '-'
74              post_json = {}
75              headers_json = {}
76              headers_json['timestamp'] = str(int(time.time()))
77              headers_json['host'] = 'norsecorp'
78              post_json['headers'] = headers_json
79              post_json['body'] = "%s;%s;%s;%s;%s;%s;%s;%s;%s;%s;%s;%s" % (
    aux_time,str(ataque['md5']),str(ataque['city']),
80
    str(ataque['countrycode']),str(ataque['latitude']),
81
    str(ataque['longitude']),str(ataque['org']),
82
    str(ataque['dport']), str(ataque['city2']),
83
    str(ataque['countrycode2']), str(ataque['latitude2']),
84
    str(ataque['longitude2']))
85              print json.dumps(post_json)
86              payload = '[' + str(json.dumps(post_json)) + ']'
87              response = requests.post('http://ip_servidor:5140', data=
    payload, headers=header)
88              print response
89              # print '\n'
90
91          #   print ataque
92          #   print '\n'
93  class Thread_main(Thread):
94      def __init__(self):
95          Thread.__init__(self)
96      def run(self):
97          global isopen, timeout
98
99          if(isopen == 0):
100             timeout = 5
101             time.sleep(5)
102             headerc = {'Accept-Encoding' : 'gzip, deflate, sdch',
103                        'Accept-Language':'pt-BR,pt;q=0.8,en-US;q=0.6,en;q
    =0.4,es;q=0.2',
104                        'Cache-Control':'no-cache',
105                        'Pragma':'no-cache',
106                        'Sec-WebSocket-Extensions':'permessage-deflate;
    client_max_window_bits',
```

```
107                         'User-Agent':'Mozilla/5.0 (X11; Linux x86_64)
    AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36
    '}
108
109             websocket.enableTrace(True)
110             websocket.setdefaulttimeout(10)
111             ws = websocket.WebSocketApp("ws://map.norsecorp.com/
    socketcluster/",header = headerc,
112                                         on_message = on_message,
113                                         on_error = on_error,
114                                         on_close = on_close)
115             ws.on_open = on_open
116             ws.run_forever()
117
118 if __name__ == "__main__":
119     writePidFile()
120     while 1:
121         thread_captura = Thread_main()
122         thread_captura.start()
123         thread_captura.join()
```

Apêndice F

Catcher para captura dos dados da empresa LookingGlass Cyber

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals
import websocket
from threading import Thread
import json
import datetime
import time
import os
import requests

timeout = 5
isopen = 0
threads = []
header = {
    'Content-Type': 'application/json; charset=UTF-8'
}

def formatTime(dt, precision=3):
    us = str(dt.microsecond)
    f = us[:precision] if len(us) > precision else us
    return "%d-%d-%d %d:%d:%d.%d" % (dt.year, dt.month, dt.day, dt.hour, dt
    .minute, dt.second, int(f))

def on_message(ws, message):
    global timeout,isopen,threads

    timeout -= 1
    if(timeout < 0):
        ws.close()
```

```
29      elif(message.find('"rid":1') != -1):
30          isopen = 1
31          print("Rid 1 recebido!");
32          ws.send('{"event":"#subscribe","data":{"channel":"global"},"cid":2}
        ');
33      elif(message.find('"rid":2') != -1):
34          print("Rid 2 recebido!");
35      elif(message == "#1"):
36          print("Ping recebido!");
37          ws.send("#2");
38      else:
39          timeout = 5
40          newthread = Thread_handler_capture(message)
41          newthread.start()
42          print "Tratando mensagem"
43          threads.append(newthread)
44
45  def on_error(ws, error):
46      print(error)
47
48  def on_close(ws):
49      global isopen,threads
50      isopen = 0
51      for thread in threads:
52          thread.join()
53      print("### closed ###")
54
55  def on_open(ws):
56      ws.send('{"event":"#handshake","data":{"authToken":null},"cid":1}')
57
58  def writePidFile():
59          pid = str(os.getpid())
60          currentFile = open("/var/run/websocket_norse_cloudera.pid", "w")
61          currentFile.write(pid)
62          currentFile.close()
63
64  class Thread_handler_capture(Thread):
65      def __init__(self, message):
66          Thread.__init__(self)
67          self.message = message
68      def run(self):
69          global header
70          message_json = json.loads(self.message)
71          for ataque in message_json['data']['data']:
72              aux_time = str(formatTime(datetime.datetime.utcnow()))
```

```
73              print '-'
74              post_json = {}
75              headers_json = {}
76              headers_json['timestamp'] = str(int(time.time()))
77              headers_json['host'] = 'norsecorp'
78              post_json['headers'] = headers_json
79              post_json['body'] = "%s;%s;%s;%s;%s;%s;%s;%s;%s;%s;%s;%s" % (
    aux_time,str(ataque['md5']),str(ataque['city']),
80
    str(ataque['countrycode']),str(ataque['latitude']),
81
    str(ataque['longitude']),str(ataque['org']),
82
    str(ataque['dport']), str(ataque['city2']),
83
    str(ataque['countrycode2']), str(ataque['latitude2']),
84
    str(ataque['longitude2']))
85              print json.dumps(post_json)
86              payload = '[' + str(json.dumps(post_json)) + ']'
87              response = requests.post('http://ip_servidor:5140', data=
    payload, headers=header)
88              print response
89              # print '\n'
90
91          #   print ataque
92          #   print '\n'
93  class Thread_main(Thread):
94      def __init__(self):
95          Thread.__init__(self)
96      def run(self):
97          global isopen, timeout
98
99          if(isopen == 0):
100             timeout = 5
101             time.sleep(5)
102             headerc = {'Accept-Encoding' : 'gzip, deflate, sdch',
103                        'Accept-Language':'pt-BR,pt;q=0.8,en-US;q=0.6,en;q
    =0.4,es;q=0.2',
104                        'Cache-Control':'no-cache',
105                        'Pragma':'no-cache',
106                        'Sec-WebSocket-Extensions':'permessage-deflate;
    client_max_window_bits',
```

```
107                         'User-Agent':'Mozilla/5.0 (X11; Linux x86_64)
    AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36
    '}
108
109             websocket.enableTrace(True)
110             websocket.setdefaulttimeout(10)
111             ws = websocket.WebSocketApp("ws://map.norsecorp.com/
    socketcluster/",header = headerc,
112                                         on_message = on_message,
113                                         on_error = on_error,
114                                         on_close = on_close)
115             ws.on_open = on_open
116             ws.run_forever()
117
118 if __name__ == "__main__":
119     writePidFile()
120     while 1:
121         thread_captura = Thread_main()
122         thread_captura.start()
123         thread_captura.join()
```

Apêndice G

Programa para processamento do dados em Python

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#Imports
from numpy.random import rand
from pylab import plot,show,savefig,figure
from numpy import vstack,array,concatenate
from scipy.cluster.vq import kmeans,vq
from pymongo import MongoClient
import random

def recupera_dados(informacao_X, informacao_Y, rand):

  #Conexao com o banco de dados
   client = MongoClient('localhost', 27017)
   database = client.norsecorpDB
   collection = database.attack
     count = collection.count()

     #Definindo um tamanho de amostra
   tamanho_amostra = 1000000

   #Definindo array com informacoes do ataque
   amostra = []

     if(rand):
       documentos = collection.find().limit(1).skip(random.randrange(count)
    /10).next()
     else:
```

```
29      #Realizando a consulta ao banco de dados
30      documentos = collection.find().limit(tamanho_amostra)
31
32    i = 0
33    for documento in documentos:
34      if i == 0:
35          #Atribui para amostra um par ordenado contendo a hora do dia e
      numero da porta atacada
36        amostra = array([[int(documento[informacao_X][11:13]), int(documento[
      informacao_Y])]])
37            i += 1
38      else:
39          #Atribui para amostra um par ordenado contendo a hora do dia e
      numero da porta atacada
40        amostra_temp = array([[int(documento[informacao_X][11:13]), int(
      documento[informacao_Y])]])
41        amostra = concatenate((amostra, amostra_temp),axis=0)
42
43 amostra = recupera_dados('timestamp','dport',0)
44
45 #Define os centroides dos grupos aplicando o metodo K-Means
46 #Neste caso dividindo somente em dois grupos
47 centroids,_ = kmeans(amostra,2)
48
49 #Calculando a distancia de cada ponto ate as duas centroides para definir a
       qual grupo aquele ponto pertence.
50 idx,_ = vq(amostra,centroids)
51
52 #Exibe o calculo da distancia
53 print idx
54
55 #Configura MatPlotlib para exibicao do grafico
56 fig = figure()
57 ax = fig.add_subplot(111)
58 ax.set_title("Horas do dia X Portas de destino")
59 ax.set_xlabel("Horas do dia")
60 ax.set_ylabel("Portas de destino")
61
62 #Projeta os ponto sobre o grafico em tela
63 ax.plot(amostra[idx==0,0],amostra[idx==0,1],'ob',
64        amostra[idx==1,0],amostra[idx==1,1],'or')
65 #Projeta os centroides sobre o grafico em tela
66 ax.plot(centroids[:,0],centroids[:,1],'sg',markersize=8)
67
68 #Salva o grafico gerado
```

```
69 savefig('teste_tcc2.png', bbox_inches='tight')
70
71 #Exibe o grafico
72 show()
```